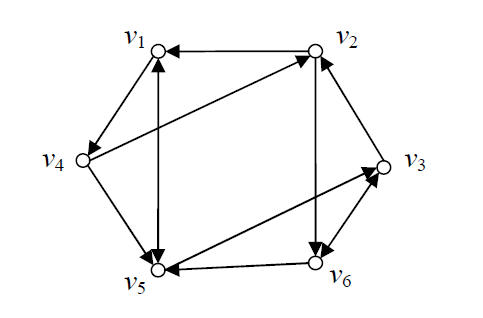
# 1 АНАЛІТИЧНИЙ ОГЛЯД ЗА ТЕМОЮ ДИПЛОМНОЇ РОБОТИ

У даному розділі розглянуто математичний опис завдання пошуку маршрутів.

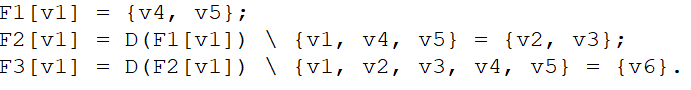
## Постановка задачі трасування маршрутів

Алгоритм отримав назву «хвильовий» черзе те, що значення індексів n які отримують вершини графу H відраховується, як шлях з стартової вершини v своєрідною хвилі, яка спрямовується за напрямком дуг. Алгоритм закінчить свою роботу коли хвиля дійде до кінцевої вершини. Величина індексу, який прийшов із хвилею, у вершині буде дорівнювати довжині знайденого маршруту. У свою чергу, для визначення маршруту потрібно з кінцевої вершини в напрямку стартової вершини помічати послідовно одну вершину зі значеннями індексу n–1, n–2, …, і так до нуля. Виходить, що вершина, індекс якої дорівнює 0 – є початковою. Але якщо було знайдено кінцеву вершину, це ще не означає що саме тей маршрут є єдиним, а вибраний режим визначається тим, як працює конкретний алгоритм системи.

Як приклад, визначаємо маршрут мінімальної длини з v1 до v6 в графі, заданому (відповідний граф представлено на рис. 1.1).

  
Рисунок 1.1 – Приклад роботи хвильового алгоритму

Завдяки алгоритму, один за одним визначаємо (1.1).

 (1.1)

Отож, v6 ∈ F3[v1], а, отже, виходячи із пункта три існує шлях з v1 до v6 є довжиною у три — це і є мінімальний шлях. Тепер знайдемо мінімальний маршрут із v1 до v6. Визначимо множину (1.2).

 (1.2)

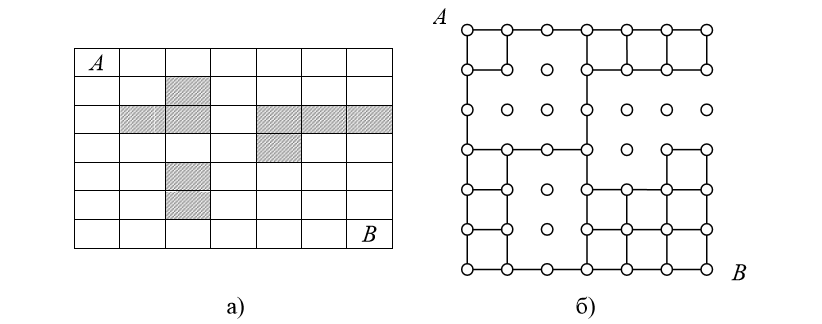
Визначимо далі множину v3 (1.3).

 (1.3)

Далі беремо будь-яку вершину із отриманого масиву, наприклад, v5. Як результат отримуэмо такий шлях (1.4):

 (1.4)

Цей мінімальний шлях із v1 до v6. А відстань шляху дорівнює трьом. Звісно, хвильовий алгоритм має застосовування не виключно серед орієнтованих, він застосовується ще для неорієнтованих. У випадку із неорієнтованими, пересування між вершинами можливі у будь-яку сторону. Метод фронту хвилі також часто пристосовується при створенні комп’ютерних ігор, наприклад, для визначення оптимального маршруту пересування гравця або іншого персонажу з однієї точки карти до іншої. Наприклад, нехай потрібно знайти найкоротший шлях з точки А до точки В на мапі, зображена на рис. 3, а. На цій мапі сірі елементи зазначено як перешкоди на маршруті, тобто в цих фрагментах мапи персонаж не може пройти. Також представимо, що маневр гравця має можливість бути реалізовано лише по вертикалі та горизонталі. Граф, відповідний данній мапі, представлено на рис. 1.2, б.

  
Рисунок 1.2 – Приклад карти території та відповідного графу

Після циклу хвильового методу отримаємо такі індекси вершин – елементів карти (рис. 1.3, а). На рис. 1.3, б зображені два зі знайдених маршрутів від вершини А у вершину В. Можна побачити, що протяжнсть знайденого шляху дорівнює 12.

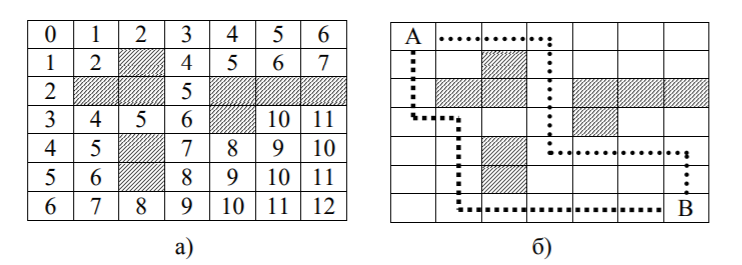


Рисунок 1.3 – Робота хвильового алгоритму для графу

Можна обновити метод знайдення найкоротшого шляху, застосувавши хвильовий алгоритм, що зробить його більш швидким, та економічним. Для цього необхідно поширювати хвилю з вхідної вершини А (перша хвиля), та фінішної В (друга хвиля). Щоб візуально розрізняти індекси хвиль — індекси нової зазначимо із додаванням штриху. Алгоритм прийнято вважати закінченним, коли дві хвилі зустрічаються (рис. 1.4).

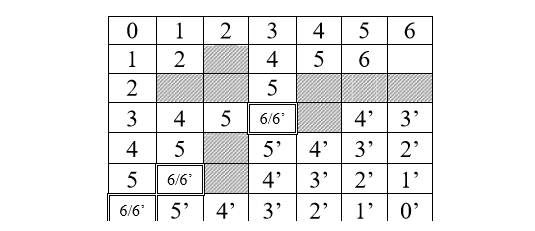


Рисунок 1.4 – Двонаправлений хвильовий алгоритм

Елементи матриці, де хвилі зустрічаються, позначені подвійною рамкою. З зіставлення помітно, що знайдені найкоротші маршрути співпадають. Однак, в цьому випадку економія є тільки один елемент, проте можна зазначити, що на складніших картах робота модифікованого методу хвилі можн буде ефективнішою за простий хвильовий алгоритм.

## Область використання трасування маршрутів

У даному підрозділі розглянуті області використання трасування маршрутів.

### 1.2.1 Використання трасування маршрутів у системах проектування печатних платформ

Трасування друкованих плат — є одним із етапів проектування радіоелектронної техніки і суть його у покроковому проектуванні структури провідників печатної плати вручну або за допомогою однієї з вже надрукованих плат.

Трасування з'єднань, найчастіше, є завершальним кроком конструкторського проектування, який полягає у визначенні геометрії, або топології провідників, що об’єднують контакти елементів проектованого прилоду.

Задача трасування — одина з найбільш трудомістких задач у автоматизації проектування печатних плат. Це певно зв’язано із різними моментами конструктивної реалізації поєднань, при цьому для кожного, при алгоритмічному розв’язання завдання застосовуються спеціальні оптимізаційні критерії. Математична точки зору каже що, задача трасування — оптимізаційна задача, яка полягає у виборі оптимального рішення з безлічі можливих варіантів.

Чисельність ймовірних варіантів в задачах трасування така, що для умовно легких систем вирішення шляхом перебору усіх можливих варіантів з'єднань — не раціональна. Таким чином методи трасування представляють інтерес. Методи які шукають не найкращий варіант трасування (це потребує гарантованого перебору та оцінки усіх можливих варіантів), а оптимальний варіант, котрий незначно поступається оптимальному, проте стане знайдений за умовно невеликий час.

Основне завдання трасування формується таким чином: за наданою схемою з'єднань прокласти потрібні провідники на платі так, щоб при цьому досягти зазначеного критерію якості трасування, ураховуваючи заздалегідь заданні обмеження. Основними обмеженнями є товщина провідників та мінімальна відстань між ними.

Методи фронту хвилі, що базуються у думках Лі, і створені Ю. Л. Зіманом і Г. Г. Рябовим. Ці алгоритми стали широко поширеня у існуючих системах автоматизації, тому що ці алгоритми дозволяють дуже просто визначити специфіку монтажу із множиною певних конструктивних перешкод. Дані методи завжди гарантують побудову траси, якщо хоча б один шлях для неї існує.

### 1.2.2 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. Підхід розробки ігрового ШІ сильно 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. С створення ігрового персонажу заснована на модулі штучного інтелекту, який пристосовано до особливостей гри: мапі, правил, а також до типа 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).

### 1.2.3 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 знаходення найкоротшого шляху; 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.

Наприклад найпростіша задача 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. Допускається піти двома путями: перший — закладати шлях «на ходу», не двивлячись на перешкоди до зіткнення з такими; 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.

### 1.2.4 Використання трасування маршрутів в задачах навігації

Алгоритми знаходження найкоротшого шляху застосовуються для знаходження маршрутів між об'єктами у програмах мап, таких як Google.

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.

## Алгоритми пошуку маршрутів

У даному підрозділі детально описано 4 найбільш використовуваних методів трасування маршрутів.

### 1.3.1 Алгоритм Дейкстри

Алгоритм Дейкстри – полягає у знаходженні найкоротшого шляху від однієї з вершин графа до всіх інших. Алгоритм працює тільки для графів без ребер з негативною вагою.

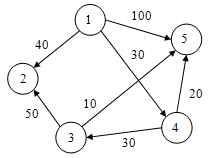
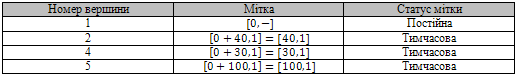
Допустимо є деякий орієнтований граф, для котрого потрібно знайти найкоротший маршрут від вершини один до всіх вершин. Для розв'язку задач такого типу можна використовувати алгоритм Дейкстри.

Рис. 1.5 — Приклад роботи алгоритму Дейкстри

Слідуючи за алгоритмом, початкова вершина (вершина під номером 1) отримує постійну мітку Алгоритму Дейкстри після чого можна переходити до першого кроку.

Крок перший: з вершини один є орієнтоване ребро до вершин два, чотри та п’ять. Далі йде обчислення даних вершин для відповідні мітки. В результаті отримаємо наступну таблицю (табл. 1.1.).

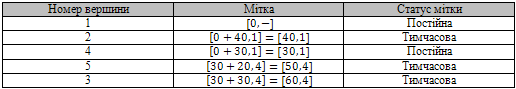
Таблиця. 1.1 — Приклад роботи алгоритму Дейкстри



Із вершин з тимчасовими мітками, беремо ту, значення відстані для якої є найменшим. В нашому випадку такою є вершина 4. Зміняємо статус цієї вершини на «постійний», далі – другий крок.

Крок Другий: з четвертої вершини є орієнтоване ребро до вершин три і п’ять. Після обчислення для них мітки, далі таблиця набуде вигляду як на таблиці 1.2.

Таблиця. 1.2 – Приклад роботи алгоритму Дейкстри

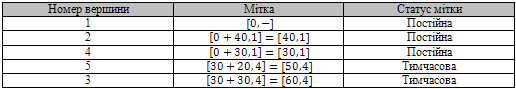


В даній таблиці, є тимчасова відмітка (100, 1) для вершини п’ять, яку отримали після попереднього крока, її замінено на нову (50, 4), також тимчасову. Виходячи з чого, можна зробити висновок що до вершини п’ять знайдено коротший шлях, який пролягає через вершину чотири. Після чого, подібно до першого кроку, вибираємо вершину відстань для якої є найменшим (зараз це вершина під номером два) після чого зміняємо статус її на «постійний».

Крок третій: якщо від вершини два немає можливості потрапити у інші вершини графа, то цей крок стає подібним до другого етапу таблицю, але статус вершини два змінено на «постійний».

Поточний статус можна побачити на таблиці 1.3.

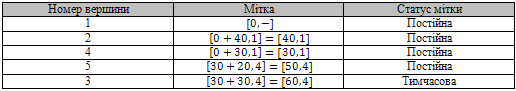
Таблиця. 1.3 — Наочний приклад роботи алгоритму Дейкстри



Серед тих вершин, статус яких не дорінює «постійний», беремо ту, для якої значення відстані є найменшим. Зараз це вершина п’ять, для якої відстать дорівнює (50, 4). Після зміни її статус на «постійний», виконується четвертий крок.

Четвертий крок: від вершини п’ять не існує орієнтованих ребер до інших вершин, тому таблиця міток залишається незмінною, і тому тільки статус вершини п’ять замінено на «постійний». Поточний статус можна побачити на таблиці 1.4.

Таблиця 1.4 – Приклад роботи алгоритму Дейкстри



На четвертому кроці, ми отримали таблицю, в якій тимчасову мітку містить тільки вершина три. І опираючись на те, що з даної вершини можна потрапити у вершини два або п’ять, статус яких дорівнює «постійний», то процес обчислень алгоритмом закінчився.

Таким чином отримано шлях найкоротшої довжини, який має початок у вершині один та обходить всі вершини графа (див. рис. 1.6).

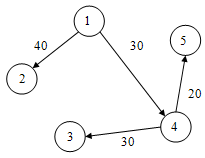


Рисунок 1.6 – Приклад роботи алгоритму Дейкстри

### 1.3.2 Алгоритм Беллмана-Форда

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

Далі метод Беллмана-Форда розглядено детальніше. Допустимо ми маємо орієнтований граф із зваженими ребрами, та абсолютна відсутність циклів, де довжина може бути від’ємною. Також допустимо, що задчача потребує визначити найкоротші шляхи від вершини «старту» до всіх інших вершин графа.

До початка виконання алгоритму всі вершини графа відмічаємо як «непройдені», а ребра як «не переглянуті». Далі кожній вершині присвоюємо число dx, яке дорівнює довжині найкоротшого шляху з вершини a у вершину x, включаючи виключно вершини, що відмічені як «пройдена». Першим кроком 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 усіх x, відмінних від a. Ще на даному кроці, вершині a присвоюємо мітку «пройдена» та припускаємо http://www.mathros.net.ua/wp-content/uploads/2019/02/algorytm_bellmana-forda14.gif ( y — це остання з вже пройдених вершин).

Для кожної вершини графа перерахувати величину таким чином : (http://www.mathros.net.ua/wp-content/uploads/2019/02/algorytm_bellmana-forda17.gif — вартість ребра ). Якщо http://www.mathros.net.ua/wp-content/uploads/2019/02/algorytm_bellmana-forda13.gif в усіх непройдених вершин, роботу методу Беллмана-Форда потрібно закінчити. Це означає що у вихідному графі відсутні шляхи з вершини у непройдені вершини. Інакше, потребується присвоїти відповідну мітку тій з вершин x, для якої dx – є найменшею серед іншиї. Та ребро, що веде до обраної на даному кроці вершині x вважається переглянутим. Після цього, примінівши http://www.mathros.net.ua/wp-content/uploads/2019/02/algorytm_bellmana-forda19.gif , ітераційний процес продовжується. Слід відмітити, що коли для якоїсь пройденої вершини x відбувається зменшення dx, то з цієї вершини і инцидентного їй вже переглянутого ребра знімаються мітки.

Робота методу Беллмана-Форда продовжується доки всі вершини графа не було відмічено як «пройдена», і доки після декількох виконань кроку номер два хочаб одне із значень dx змінилося.

### 1.3.3 Алгоритм пошуку A \*

Алгоритм пошуку A \* — використовується для знаходження маршруту з найменшою вартістю від однієї вершини (початкової) до іншої (цільової, кінцевої), використовуючи алгоритм пошуку по першому найкращому збігу на графі.

Алгоритм працює з допоміжною функцією (евристикою), щою скоректувати напрям пошуку та для зменшення тривалісті виконання. Метод має перевагу в тому, що завжди знаходить оптимальний шлях, якщо цей розв’язок існує.

Алгоритм А\* першим чином починає відвідувати вершини, які з більшою ймовірністю ведуть до шляху із мінімальною довжиною. Для розпізнавання таких вершин, до кожної відомої вершини x підставляється значення f(x), яке буде дорівнювати довжині найменшого шляху від початкової вершини до кінцевої, який пролягає через вершину х. В першу чергу обирають вершини із найменшим значенням f.

Функція f(x) для вершини x визначається так (1.3):

 (1.3)

де:

* g(x) є функцією, яка повертає вартість шляху від вершини «старт» до x;
* h(x) це евристична функція, яка повертає вартість шляху від x до «фініш»;

Але, використана евристична функція не повинна дати підвищену оцінку вартості шляху. Прикладом оцінки може служити пряма лінія, а загальний шлях не може бути коротшим за цю лінію. Приклад роботи методу можна побачити на рис. 1.7.

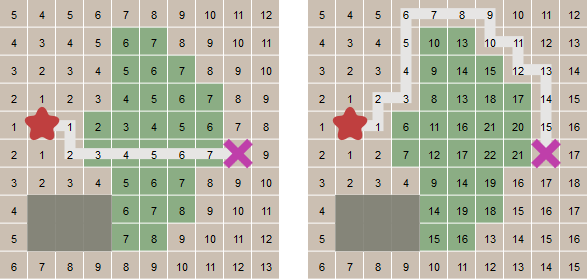


Рисунок 1.7 – Приклад роботи алгоритму пошуку A \*

Під час роботи алгоритм поділяє вершини на три класи:

* невідомі вершини: вершини, що ще не були знайдені. Тобто, шлях ще невідомий до них. Із самого початку роботи методу всі, окрім початкової, вершини відносяться до цього класу.
* відомі вершини: вже існує відомий (мабудь неоптимальний) маршрут до цих вершин. Всі відомі вершини та значеннями f зберігаються у списку. Від цього масиву вибирають ті вершини, які є найперспективнішими. Хід реалізації цього масиву має суттєвий вплив на кінцеву швидкість алгоритму, і зазвичай, виглядає як пріоритетна черга. На початку роботи алгоритму є тільки одна вершина з відомих – початкова вершина.
* повністю досліджені вершини: до цього списку відносять ті вершини, до яких найкоротший шлях вже відомо. При додавання до списку повністю досліджених вершин, вони потрапляють до замкнутого списку, це допомагає запобігти багаторазовому дослідженню вершин які вже дослідили. На початку роботи алгоритму список повністю досліджених вершин є порожнім.

Кожна вже відома або досліджена вершина має вказувати на інші вершини, які ведуть о неї. Через це, можна пройти одним шляхом від цієї вершини до початкової.

Коли вершина x буде повністю досліджена, сусідня до неї вершини увійдуть до масиву вже відомих вершин, а ця вершина буде додана до списку вже досліджених. Указателі на минулу вершину передають на x. Сусідні елементи, котрі вже є у списку повністю досліджених вершин, не попадають до масиву вже відомих, але не дослідженних. Зворотні вказівники не змінюються. Сусідні елементи, які вже знаходяться в списку відомих — оновлюються, коли новий маршрут коротший за попередній.

Коли фінішна вершина оказується у масиві повністю досліджених вершин — алгоритм зупиняється. Маршрут, який було знайдено відтворюється через указателі на попередню вершину. Якщо список відомих вершин порожній — алгоритм припиняє пошук. Це означає що шлях не можливо прокласти.

Відтворення завдяки зворотнім вказівникам починається з «фінішної» вершини та двигається до «стартової». Для отримання шляху в правильному напрямку, зі «стартової» вершини до «фінішної», в умовах задачі їх слід переставити місцями. А коли шукається маршрут із «фінішної» вершини, знайдений список починатиметься із «стартової» та буде вести до «фінішної» врешин.

## Висновки

У цьому 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.