**НАУКОВА РОБОТА**

**на тему:**

**«МОДЕЛЬ ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ МАРШРУТІВ»**

**під шифром «route optim»**

**АНОТАЦІЯ**

Побудова оптимальних транспортних маршрутів для будь-якої компанії дуже актуальна проблема, розв’язання якої надає можливості для скорочення часу та матеріальних ресурсів на виконання проекту, водночас розвантажує транспортні шляхи міста.

В роботі розглядаються та аналізуються існуючи системи побудови оптимального маршруту, порівнюються їх технічні вимоги, функціональні можливості. Розглядається математична модель транспортної мережі та алгоритми пошуку оптимального маршруту. Представлена програмна система, яка реалізує розроблені методи.

Наукова робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатку. Зміст роботи викладено на 29 сторінках друкованого тексту, включаючи 14 рисунків і 2 таблиці.

Наведені результати можуть бути використані при створені систем пошуку оптимального маршруту на підприємствах міста, а також в зонах бойових дій при виконанні розвідувальних та рятувальних операцій на відкритих місцевостях.

ЗМІСТ

[ВСТУП 4](#_Toc64027537)

[РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ПОБУДОВИ ОПТИМАЛЬНИХ ШЛЯХІВ У ДИНАМІЧНИХ МЕРЕЖАХ 7](#_Toc64027538)

[1.1. Теоретичні основи моделі для побудови оптимальних маршрутів 7](#_Toc64027539)

[1.2. Порівняльний аналіз існуючих сервісів 8](#_Toc64027540)

[1.3. Класифікація математичних підходів для побудови оптимальних маршрутів 10](#_Toc64027541)

[*1.3.1. Базові підходи на основі алгоритму Дейкстри* 11](#_Toc64027542)

[*1.3.2. Підходи на основі динамічного програмування* 13](#_Toc64027543)

[*1.3.3. Ієрархічні підходи пошуку найкоротшого шляху* 16](#_Toc64027544)

[РОЗДІЛ 2. МОДЕЛЮВАННЯ ЗАДАЧІ ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ ДЛЯ ПОБУДОВИ ОПТИМАЛЬНОГО МАРШРУТУ 18](#_Toc64027545)

[2.1. Математична модель оптимізації маршруту на основі графового подання 18](#_Toc64027546)

[2.2. Особливості математичної моделі оптимального маршруту підприємства «Співдружність» 21](#_Toc64027547)

[РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАДАЧІ ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ МАРШРУТІВ 24](#_Toc64027548)

[3.1. Вибір IDE для розробки системи оптимізації транспортних маршрутів 24](#_Toc64027549)

[3.2. Оптимізація транспортних маршрутів підприємства «Співдружність» 25](#_Toc64027550)

[ВИСНОВОК 28](#_Toc64027551)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 29](#_Toc64027552)

[ДОДАТОК А 31](#_Toc64027553)

# ВСТУП

**Актуальність теми дослідження.** В динамічних мережах, задача знаходження оптимальних маршрутів, залежних від часу, застосовується в різних сферах діяльності людини, а саме в таких областях як: логістиці – оптимізація доставки товарів; складання розкладів для різних видів транспорту; для створенні маршрутів в області туризму; навіть для оборонної сфери, с точки зору планування різних завдань з оптимальним маршрутом тощо.

Інтерес до цього обумовлено тим, що транспортна логістика може збільшити свою виручку за допомогою оптимізації інфраструктури транспорту.

Основоположні завдання, з якими стикається транспортна логістика – це розробка оптимальних маршрутів для перевезень та вибір вигідного місця для розташування логістичного об’єкта.

У сучасному світі незамінну роботу виконують вантажоперевезення, які доставляють різноманітні вантажі та кореспонденцію. На даний час, схожу незамінну роботу, виконують служби кур’єрської доставки, які перевозять найрізноманітніші вантажі – починаючи з доставки різної кореспонденції до перевезення різногабаритних предметів безпосередньо адресату. Доставка до дверей клієнта – це основний принцип роботи кур’єрської служби.

Ще одним невід’ємним фактором в житті сучасного суспільства став розвиток Інтернет-магазинів, які с кожним днем набувають популярності. Це спровокувало зростанню у населені затребуваності кур’єрських послуг.

Для будь-якої компанії, яка займається послугами в сфері логістики, існують певні основні задачі автоматизації. По-перше, це побудова оптимальних шляхів для перевезень, по-друге, оцінка витрачених на доставку вантажу ресурсів.

Також став невпинно розвиватися клас задач пошуку оптимальних туристичних маршрутів, які включають задачі пошуку найкоротшого шляху через задані вершини, дослідження пошуку шляхів для комбінації різних видів транспорту та розв’язання задачі комівояжера для динамічного маршруту між заданими пам’ятками.

Вирішення цих гострих завдань, які піднімається в даній роботі, безсумнівно, говорить про її актуальність.

***Метою***дослідження є підвищення ефективності пошуку маршрутів у транспортних мережах, удосконалення моделі оптимізації транспортних шляхів.

***Об’єктом дослідження*** є процес побудови оптимальних шляхів. В роботі увага приділяється задачам пошуку найкоротших шляхів для оптимізації системи транспортних маршрутів, а саме знаходженню самого вигідного шляху, що проходить через вказані міста хоча б по одному разу з поверненням у вихідне місто, задачу, що проходить із заданої точки до інших точок та ін.

***Предметом дослідження*** є моделі, основні методи та підходи побудови оптимальних транспортних шляхів.

Для досягнення поставленої мети в дослідженні вирішуються такі завдання:

* проаналізувати теоретичні аспекти побудови оптимальних транспортних шляхів у динамічних мережах, які включають моделі для побудови оптимальних маршрутів, зробити порівняльний аналіз існуючих сервісів та класифікація основних математичних підходів знаходженню оптимальних шляхів;
* удосконалити математичну модель оптимізації транспортних маршрутів на основі графового подання для підприємства «Співдружність»;
* розробити програмний продукт для реалізації вищезазначеної моделі.

**Методи дослідження.** Теоретичною основою дослідження є фундаментальні положення дискретної математики, теорії графів та системного аналізу.

**Наукова новизна** роботи полягає в удосконаленні моделі для оптимізації процесів транспортної логістики підприємства.

Наукову роботу виконано відповідно до тематики комплексної теми кафедри математичних методів та системного аналізу Маріупольського державного університету «Розробка програмного та науково-методичного забезпечення для практичної підготовки здобувачів вищої освіти» (номер держреєстрації 0118U0000551), в рамках якої автором розвинуто науково-методичні підходи до математичного моделювання, а саме удосконалено модель оптимізації транспортних маршрутів, розглянуто та удосконалено методи підвищення ефективності маршрутів у транспортних мережах та виконано програмну реалізацію цієї моделі.

Результати наукової роботи впроваджено в начальний процес Маріупольського державного університету і використано при викладанні дисципліни «Теорія прийняття рішень».

**Апробація результатів дослідження.** Основні результати дослідження доповідались автором та отримали схвалення на Декаді студентської науки – 2021 (м. Маріуполь, 2021р.).

**Публікації.** За результатами виконаного дослідження подано до друку тези доповіді у збірнику тез доповідей студентів за результатами Декади студентської науки – 2021.

**Обсяг та структура роботи.** Наукова робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатку. Зміст роботи викладено на 29 сторінках друкованого тексту, включаючи 14 рисунків і 2 таблиці.

# РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ПОБУДОВИ ОПТИМАЛЬНИХ ШЛЯХІВ У ДИНАМІЧНИХ МЕРЕЖАХ

# Теоретичні основи моделі для побудови оптимальних маршрутів

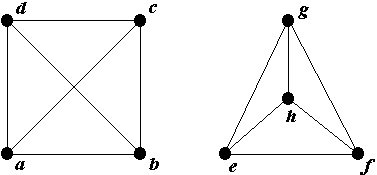
*Математичними моделями* називають комплекс математичних залежностей і логічних виразів, що відображають суттєві характеристики досліджуваного явища (процесу). Математичні залежності встановлюють залежність між відомими і шуканими величинами [1].

Модель – це відображення системи, що досліджується, або деяких явищ, що нас цікавлять (процесів, подій). Моделі використовуються в цілях управління і прогнозування (передбачення) можливої ефективності системи в разі зміни її характеристик. Вони дозволяють пояснити явища, що нас цікавлять, і виявити взаємозв'язок окремих явищ, а також замінити дорогі та складні випробування систем в реальній обстановці. Моделювання здійснюється з метою удосконалення системи, а також для ознайомлення і навчання персоналу з системами та ситуаціями, які важко здійснити в реальних умовах. За допомогою моделі можна перевіряти або демонструвати нові ідеї, отримувати кількісну оцінку при різних варіантах управління.

Для побудови оптимальних маршрутів можна скористатися потужним засобом дискретної математики – математичним апаратом теорії графів, який дає змогу ефективно розв’язувати задачі оптимізації [2].

Теорія графів є простим, доступним і потужним засобом вирішення широкого кола питань із різних галузей, що дає змогу використовувати її для математичного моделювання задач на знаходження оптимального шляху.

Велику кількість реальних ситуацій зручно моделювати за допомогою рисунків, що складаються з точок (вершин), які представляють основні елементи ситуації, та ліній (ребер), що з’єднують певні пари цих вершин і описують зв’язки між ними. Такі рисунки у математичній літературі називають графами (рис.1.1).



*Рис. 1.1. Приклади графів.*

З графами, самі того не помічаючи, ми стикаємося постійно. Найпростішим прикладом графів є схема доріг чи ліній метрополітену. У ній точкам відповідають населені пункти (станції метро), а лініям, що їх сполучають – шляхи руху автомобілів (поїздів).

Графи є зручним засобом опису зв’язків між об’єктами. Зокрема, розглядаючи граф, що зображує мережу доріг між населеними пунктами, можна визначити маршрут проїзду від пункту А до пункту В. Якщо таких маршрутів виявиться кілька, можна вибирати з них оптимальний, наприклад, найкоротший або найбезпечніший.

Для вирішення подібних завдань вибору потрібно проводити певні дослідження графів і мати певні алгоритми їх обробки. На сьогоднішній день існує досить велика кількість готових алгоритмів дослідження графів, які моделюють найрізноманітніші завдання з багатьох сфер людської діяльності. З такими підходами ознайомимося пізніше.

# Порівняльний аналіз існуючих сервісів

На даний час існує багато різних сервісів для оптимізації маршруту: мапи, за допомогою яких можна прокласти маршрути; спеціальні системи для управління транспортом; онлайн-гіди тощо.

В цьому підрозділі розглянемо та проаналізуємо декілька таких сервісів, підкреслимо їхні переваги та недоліки.

***Сервіс «Мурашина логістика»***

«Мурашина логістика» – хмарна система управління транспортом, за допомогою якої можна автоматично планувати маршрути, оцінити рентабельність точок доставки, GPS контроль маршрутів руху, переглядати аналітичну звітність [3].

Ефективне управління перевезеннями передбачає вирішення двох ключових задач – розрахунок оптимальних маршрутів та контроль виконання маршрутів.

За допомогою цієї програми, користувач зможе автоматизувати транспортну систему, а саме: планувати оптимальні маршрути, моніторинг маршрутів, повний контроль видаткової частини, план-факт аналіз.

Якщо людина складає маршрут вручну, то це вимагає значної витрати часу та відмінних знань місцевості. Але за допомогою сервісу, складання транспортних маршрутів займає всього декілька хвилин і не вимагає особливих вмінь. Мапа для логістики, що використовується в сервісі надає наочне уявлення всієї поточної оперативної ситуації на маршрутах. Простий, інтуїтивно зрозумілий інтерфейс, дозволяє будь-якому користувачеві ПК виконати розрахунок маршрутів доставки, навіть якщо раніше він не мав досвіду роботи в аналогічних програмах. Програма легко інтегрується з обліковими системами замовника.

При плануванні маршрутів система враховує понад 60 параметрів. Розрахунок маршрутів для 1000 адресних пунктів система виконує за 18 хвилин.

Переваги системи:

* OpenStreetMap – найактуальніші мапи. Легко розширюють географію бізнесу користувача.
* Надійність – зберігає дані в захищеному хмарному сховищі.
* Швидкий старт – освоєння сервісу протягом години.
* Пробний період – можна оцінити роботу сервіса безкоштовно протягом 30 днів.
* Технічна підтримка – можна звернутися за технічною підтримкою цілодобово, без вихідних.

Головні недоліки – сервіс є платним (не враховуючи пробний період) та потребує авторизації.

***Сервіс «EasyWay»***

EasyWay mobile – маршрути громадського транспорту в телефоні користувача [4].

Сервіс надається в країнах Європи та Азії.

Можливості: Offline-режим роботи; пошук оптимального шляху від точки А в точку В; повний список актуальних маршрутів з промальовуванням на карті в кожному із запропонованих міст (схема і інтервал руху, час роботи, вартість проїзду); GPS-трекінг пересування; GPS-дані громадського транспорту (певні міста); визначення напрямку руху на базі GPS-даних і розрахунок часу прибуття транспорту на зупинку; інформація про транспорт, адаптований для людей з обмеженою мобільністю; багатомовний інтерфейс.

Недоліки – платний контент та сервіс можна встановити не для всіх версій Android .

***Сервіс «GoogleMaps»***

«GoogleMaps» – програма для планування маршрутів та знаходженню нових цікавих місць по всьому світі [5].

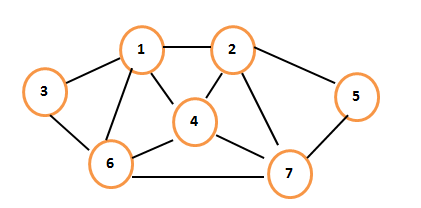
Сервіс являє собою карту та супутникові знімки планети Земля. За допомогою цієї програми користувач зможе відслідкувати дорожню ситуацію в режимі реального часу і йому буде запропоновано найкращий маршрут, дізнатися як виглядає місце, до якого користувач направляється, до того, як він туди добереться, можна навіть створити власну мапу та поділитися нею з іншими.

Сервіс має багато переваг, він являється одним із найпопулярніших програм в світі, тому ним користуються мільярди людей.

# Класифікація математичних підходів для побудови оптимальних маршрутів

*Задача пошуку найкоротшого шляху* є класичною задачею комбінаторної оптимізації, яка полягає у мінімізації суми ваг ребер, що складають маршрут мiж двома заданими вершинами на зваженому орієнтованому графі. Широке застосування задач у рiзних галузях людської діяльності сприяє її активному вивченню i дослідженню нових алгоритмів та підходів. У пошукових системах в режимі реального часу нові маршрути повинні бути визначені протягом адекватного часу після запиту користувача Традиційні методи пошуку оптимального найкоротшого шляху часто не можуть бути використані для застосування в реальному часі через свою обчислювальну складність [6].

Прикладом може бути знаходження найкоротшого шляху між двома пунктами на дорожній мапі; в цьому випадку, вершинами є пункти, а ребрами – дорога між ними зі своєю відстанню(вагами), рівними часу, необхідному для подолання цієї відстані. Така задача іноді згадується як *здача про найкоротший шлях між парою вершин.*



*Рис. 1.2. Граф для пошуку найкоротшого шляху*

Для розв’язання таких видів задач, а саме про знаходження найкоротшого маршруту, розглянемо основні підходи їх вирішення, такі як: алгоритм Дейкстри, динамічне програмування, ієрархічні підходи.

## *1.3.1. Базові підходи на основі алгоритму Дейкстри*

Для реалізації задачі про найкоротший шлях застосовують підходи на основі алгоритму Дейкстри. Цей алгоритм на графах, який був винайдений нідерландським вченим Е. Дейкстрою в 1959 році, знаходить найкоротшу відстань від однієї вершини графа до всіх інших. Його часто називають алгоритмом розташування міток. Недолік даного алгоритму в тому, що він буде функціонувати неналежним чином, якщо граф матиме дуги негативної ваги [7,11].

У процесі роботи цього алгоритму вузлам мережі *xіS* приписують числа (мітки) *d(хі)*, які служать оцінкою довжини (ваги) найкоротшого шляху від вершини *s* до вершини *xі*, що має вагу *d(xi)*. Мітки можуть бути лише тимчасовими або постійними. Перетворення мітки в постійну означає, що найкоротший шлях від вершини *s* до відповідної вершини знайдено. Сам алгоритм складається із двох етапів. На першому знаходиться довжина найкоротшого шляху,на другому вибудовується сам шлях від вершини *s* до кінцевої вершини *t*.

*Основні етапи рішення задачі**алгоритму Дейкстри:*

*1 етап*.Знаходження довжини найкоротшого шляху.

*Перший крок.* Присвоєння вершинам початкових міток.

Припустимо, що *d(s)=0\*і* будемо вважати цю мітку постійною (постійні мітки позначаються вгорі зірочкою). Для інших вершин *хіS, xi ≠ S* припускаємо *d(xi) = ∞* і вважаємо ці мітки тимчасовими. Нехай =*s, –* позначення поточної мітки.

*Другий крок.*Заміна міток.

Для кожної вершини *xi*із тимчасовою міткою, що безпосередньо слідує за вершиною , змінюємо її мітку у відповідності з наступним правилом:

*dнов.(xi)=min.*

*Третій крок.* Перетворення мітки із тимчасової в постійну.

Із всіх вершин з тимчасовими мітками вибираємо вершину з найменшим значенням мітки

*d()=min.*

Перетворюємо цю мітку в постійну и припускаємо, що =.

*Четвертий крок.* Перевірка на завершення першого етапу.

Якщо *,* то*d() –* довжина найкоротшого шляху від *s до t.* В іншому випадку відбувається повернення до *другого кроку*.

*2 етап.*Побудова найкоротшого шляху.

*П’ятий крок.* Поступовий пошук дуг найкоротшого шляху.

Серед вершин, що безпосередньо передують вершині з постійними мітками, знаходимо вершину , що задовольняє співвідношенню

*d()=d()+.*

Додаємо дугу до шуканого шляху і припускаємо *.*

*Шостий крок.* Перевірка на завершення другого етапу.

Якщо , то найкоротший шлях знайдений – його утворює послідовність дуг, отриманих на *п’ятому кроці* і розміщених у зворотному порядку. В іншому випадку повертаємося до *п’ятого кроку.*

Алгоритм Дейкстри є оптимальним для пошуку шляху практично в будь-яких графах, але він має одне обмеження. Алгоритм Дейкстри непридатний для графів, що містять ребра з негативним вагою. Для пошуку найкоротшого шляху в таких графах зазвичай використовують алгоритм Форда-Беллмана.

## *1.3.2. Підходи на основі динамічного програмування*

Динамічне програмування являє собою інструментарій, що дозволяє здійснювати планування багатокрокових керованих процесів, а також процесів, які розвиваються в часі [8].

До задач, що розв'язуються алгоритмами динамічного програмування відносяться такі, які пов'язані з оптимальним розподілом капіталовкладень, розподілом продукції між різними регіонами, *визначенням найкоротшого шляху* завезення товарів споживачам, завданнями по заміні обладнання, оптимального управління запасами тощо.

*Динамічне програмування* – це метод оптимізації багатокрокових або багатоетапних процесів, критерій ефективності яких має властивість адитивності (тобто загальний дохід процесу дорівнює сумі локальних доходів на окремих етапах). В задачах динамічного програмування критерій ефективності називається доходом. Дані процеси керовані, і від правильного вибору управління залежить величина доходу.

Показник ефективності завдання в цілому позначимо через *W*, а показники ефективності на окремих кроках – через . *W* має властивість адитивності, тобто .

Змінна , від якої залежить виграш на *i*-му кроці *і*, отже, виграш в цілому, називаєтьсякроковим управлінням, . Управлінням процесу в цілому x називається послідовність крокових управлінь (вектор управлінь) .

Оптимальне управління – це значення управління *x*, при якому значення є максимальним (або мінімальним, якщо потрібно зменшити програш). ,  де *X* – область допустимих управлінь.

Оптимальне управління в цілому визначається послідовністю оптимальних крокових управлінь .

В основі методу динамічного програмування лежить принцип оптимальності Беллмана, що формулюється наступним чином: управління на кожному кроці треба вибирати так, щоб оптимальною була сума виграшів на всіх кроків, які залишилися, включаючи виграш на даному кроці.

Можна визначити наступні *основні етапи складання математичної моделі задачі динамічного програмування*.

1. Розбиття задачі на кроки (етапи). Крок не повинен бути занадто дрібним, щоб не проводити зайвих розрахунків і не повинен бути занадто великим, щоб не ускладнювати процес крокової оптимізації.
2. Вибір змінних, що характеризують стан *s* модельованого процесу перед кожним кроком, і виявлення обмежень. В якості таких змінних слід брати фактори, що представляють інтерес для дослідника, наприклад річний прибуток при плануванні діяльності підприємства.
3. Визначення множини крокових управлінь і обмежень, тобто області допустимих управлінь *X*.
4. Визначення виграшу , який принесе на *i*-му кроці управління , якщо система перед цим перебувала в стані *s*.
5. Визначення стану , в яке переходить система зі стану *s* під впливом управління ), де *fi*-функція переходу на *i*-му кроці зі стану *s* в стан .
6. Складання рівняння, що визначає умовний оптимальний виграш на останньому кроці, для стану s модельованого процесу
7. Складання основного функціонального рівняння динамічного програмування, що визначає умовний оптимальний виграш для даного стану *s* з *i*-го кроку і до кінця процесу через вже відомий оптимальний виграш –го кроку і до кінця:

Після того як виконані пункти 1-7, і математична модель складена, приступають до її розрахунку. Зазначимо *основні етапи рішення задачі динамічного програмування.*

1. Визначення множини можливих станів для останнього кроку.
2. Проведення умовної оптимізації для кожного стану *s*, що належить на останньому *m*-му кроці за формулою і визначення умовного оптимального управління *x(s).*
3. Визначення множини можливих станів для *i*-го кроку, *i = 2, 3, ... m-1*.
4. Проведення умовної оптимізації для *i*-го кроку, *i = 2, 3, ... m-1* для кожного стану *s*, що належить , за формулою і визначення умовного оптимального управління *.*
5. Визначення початкового стану системи , оптимального виграшу і оптимального управління за формулою при *i = 1*. Це є оптимальний виграш для всієї задачі .
6. Проведення безумовної оптимізації управління. Для проведення безумовної оптимізації необхідно знайдене на першому кроці оптимальне управління підставити у формулу і визначити наступний стан системи . Для зміненого стану знайти оптимальне управління , підставити у формулу і т.д. Для *i*-го стану знайти і і т.д.

Динамічне програмування дає досить ефективний метод послідовного прийняття рішень про оптимізацію роботи системи за допомогою рішення серії пов'язаних між собою однокрокових завдань Це дозволяє говорити про багатокроковому процесі прийняття рішення. Таким способом можуть бути оптимізовані динамічні системи, для яких чинник часу відіграє істотну роль. Однак специфічний спосіб вирішення завдань динамічного програмування, який оперує рекурентними співвідношеннями, дозволяє оптимізувати не тільки реальні динамічні системи.

Таким чином, підхід, реалізований в динамічному програмуванні, укладений в заміні рішення вихідної багатовимірної задачі послідовністю задач меншої розмірності.

Однією з найбільш відомих сфер застосувань методів динамічного програмування є така область математичної економіки, як теорія управління запасами. Її предметом є розробка та дослідження математичних моделей систем, що займають проміжне положення між джерелами (виробниками) тих чи інших ресурсів та їх споживачами. До динамічних завданням управління запасами відносяться: завдання пошуку оптимального маршруту доставки товару, завдання оптимального забезпечення попиту споживачів на деякий продукт на n періодів, завдання оптимального розподілу коштів на реконструкцію і модернізацію між декількома об'єктами і інші завдання.

## *1.3.3. Ієрархічні підходи пошуку найкоротшого шляху*

Ієрархічні методи використовують властивості ієрархічності дорожньої мережі (міжнародні шосе, регіональні, міські і т.д.) [9]. Зазвичай, для великих відстаней дороги класифікують за їх важливістю або пропускною спроможністю, тобто магістралі є більш вагомими, а сільські дороги найменш вагомі. Ці властивості можна перенести на граф з призначенням ступеню суттєвості вершин і дуг. Тоді модифікація алгоритму Дейкстри може ігнорувати вершини з низьким ступенем важливості, якщо вони віддалені від початкової і кінцевої вершин. Проте, такий підхід не є достатньо гнучким (наприклад, якщо використовується метрика часу подорожі, то важливість вершин необхідно постійно перераховувати з врахуванням заторів). Тому було запропоновано підхід Reach, який гарантує оптимальність отриманих шляхів, може бути застосований з іншими евристиками та добре адаптується до динамічних умов.

Охоплення (reach) вершини – довжина найкоротшого шляху, якому вона належить. Нехай *P* – найкоротший шлях між початковим і кінцевим пунктами, який містить пункт *u* . Охоплення *r(u,P)* вершини *u* відносно *P* визначається як *min{g(s,u),g(u,d)}.* Глобальне охоплення вершини *u* у графі *G* – максимальне охоплення *u* над усіма найкоротшими шляхами, які містять *u:r(u)=max r(u,P).* Інтуїтивно зрозуміло, що вершина з великим охопленням розташована близько до середини деякого довгого найкоротшого шляху; вершини на автомагістралях мають велике охоплення, а локальні дороги – мале. Під час запиту охоплення допомагають відсікати простір пошуку. Значення охоплень розраховуються на етапі попередньої обробки. Розрахунок точних охоплень вимагає обчислення найкоротших шляхів для усіх вершин графу, що є надто ресурсоємним для великих реальних дорожніх мереж. Гутман показав, що такі охоплення можна обчислювати швидше, використовуючи часткові дерева найкоротших шляхів. Гольдберг та ін. показали, що використання міток найкоротшого шляху дозволяє зменшити охоплення для більшості вершин і значно прискорити алгоритм, адаптуючи його до застосування у реальних задачах.

Підхід *скорочення ієрархій* (Contraction Hierarchies) полягає у використанні концепції міток (shortcuts або «скорочуючих ребер») найкоротших шляхів. Мітка найкоротшого шляху – це дуга *(u,v)*, яка не обов’язково існує у початковому графі, але позначає найкоротший шлях з *u* до *v* в графі *G* . Метою є розширення графу *G* мітками таким чином, щоб під час запитів маршрутів для великих відстаней використовувати ці мітки і скорочувати простір пошуку. На етапі попередньої обробки метод евристично впорядковує вершини згідно їх важливості, а потім скорочує в оберненому порядку. Для скорочення вершини *v* вона тимчасово вилучається з графу *G* , а між кожною парою її сусідніх вершин *u* і *w* додаються мітки найкоротшого шляху в тому випадку, якщо найкоротший шлях з *u* в *w* єдиний і містить *v*. Під часу запиту пошуку шляху виконується двонаправлений пошук з *s* в *d* на скороченому графі *G*.

# РОЗДІЛ 2. МОДЕЛЮВАННЯ ЗАДАЧІ ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ ДЛЯ ПОБУДОВИ ОПТИМАЛЬНОГО МАРШРУТУ

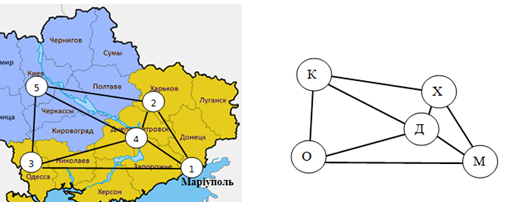
# 2.1. Математична модель оптимізації маршруту на основі графового подання

Розв’яжемо за допомогою алгоритму Дейкстри задачу, що полягає у знаходженні самого вигідного транспортного маршруту, що проходить із заданої точки до інших точок.

*Задача.*

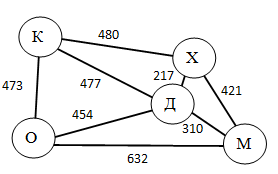
Розглянемо задачу на знаходження оптимального шляху і відстані від Маріуполя (М), до міст, які розташовані в різних областях України: Харків (Х), Одеса (О), Дніпро (Д), Київ (К).***.***

Створимо математичну модель задачі у вигляді графу (рис. 2.1.1.).

**

*Рис. 2.1.1. Математична модель задачі*

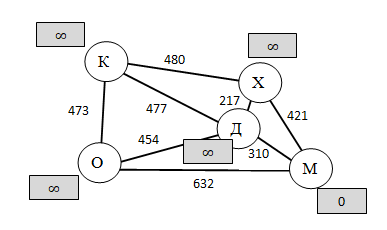
Нехай потрібно знайти найкоротшу відстань від М вершини до всіх інших. Кружками позначені вершини, лініями – шляхи між ними (ребра графа). В кружках позначені номера вершин, над ребрами їх вага – довжина шляху (рис. 2.1.2).

**

*Рис. 2.1.2.*

*Ініціалізація*

Присвоїмо мітки всім вершинам. За алгоритмом Дейкстри для нашого прикладу мінімальну мітку має вершина М., тобто дорівнює 0 (рис. 2.1.3). Мітки інших вершин – недосяжно велике число (в ідеалі - безкінченність). Це означає, що відстані від вершини 1 до інших вершин поки-що невідомі. Всі вершини графа помічаються як не відвідані.

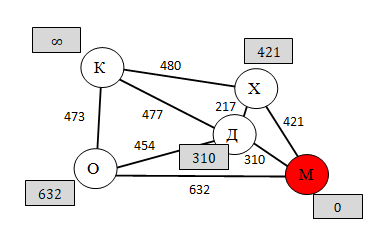
**

*Рис. 2.1.3.*

*Перший крок*

Мінімальну мітку має вершина М. Її сусіди – Д, Х і О. Обходимо вершини сусідів по черзі. Перший сусід вершини М – вершина Д, бо довжина шляху до неї мінімальна. Довжина шляху до неї через вершину М дорівнює сумі найкоротшої відстані до вершини М, значенню її мітки, і довжини ребра, що йде із М в Д, тобто 0+310=310, тому нова мітка третьої вершини дорівнює 310 (рис. 2.1.4). Це менше ніж ∞, тому нова мітка вершини Д дорівнює 310.

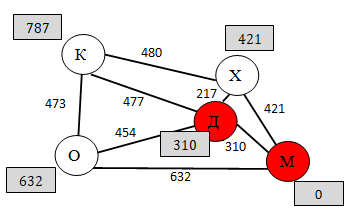
Аналогічно знаходимо довжину шляхів до інших сусідів (вершини Х, О). Всі сусіди вершини M перевірені. Поточна мінімальна відстань вважається остаточною і перегляду не підлягає. Вершина М відмічається як відвідана.



*Рис. 2.1.4.*

*Другий крок*

Перший крок алгоритму повторюється, знову знаходимо «найблищю» із не відвіданих вершин (рис. 2.1.5). Це вершина Д з міткою 310. Знову намагаємося зменшити мітки сусідів вибраної вершини, намагаючись пройти в них через вершину Д. Сусідами вершини Д є вершини М, Х, О, К. Вершина M вже відвідана. Наступний сусід вершини Д – вершина К, так як має мінімальну мітку із вершин, відмічених як не відвідані. Якщо іти до неї через Д, то довжина такого шляху дорівнює 787 (310+477=787). Це менше ніж ∞, тому нова мітка вершини К дорівнює 787.

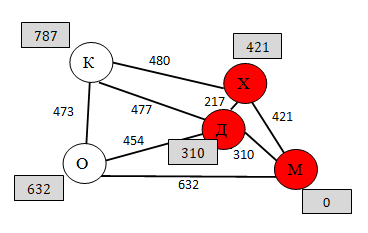


*Рис. 2.1.5.*

Ще один сусід вершини Д – вершина Х. Якщо іти до неї через вершину Д, то довжина такого шляху дорівнює 527 (310+217=527). Оскільки 527>421, то поточна мітка вершини Х не міняється. Аналогічно встановимо, що мітка вершини O залишиться 632.

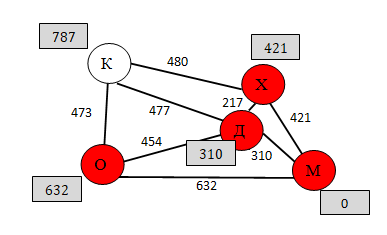
Всі сусіди вершини Д переглянуті, помічаємо її як відвідану.

*Третій крок(рис. 2.1.6.)*



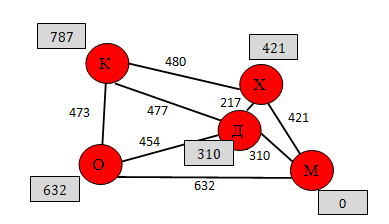
*Рис. 2.1.6.*

*Четвертий крок(рис. 2.1.7.)*



*Рис. 2.1.7.*

*П’ятий крок(рис. 2.1.8.)*



*Рис. 2.1.8.*

*Результат*: найкоротший шлях від вершини М становить

М→ Д – 310 км

М→ Х –421 км

М→ О – 632 км

М→ К – 787 км (М→ Д → К)

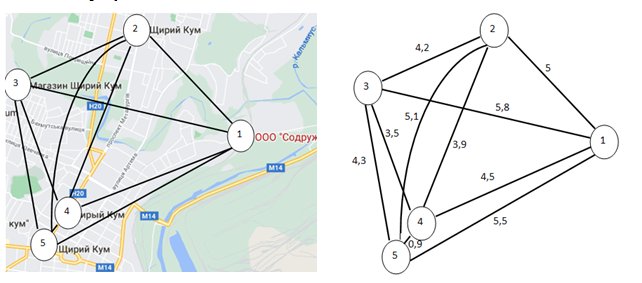
# 2.2. Особливості математичної моделі оптимального маршруту підприємства «Співдружність»

Задача комівояжера – одна з найвідоміших задач комбінаторної оптимізації, що полягає у знаходженні самого вигідного маршруту, що проходить через вказані міста хоча б по одному разу з поверненням у вихідне місто.

Теорія графів забезпечує просте, але ефективне рішення поставленої задачі. Тож розглянемо авторську задачу прийняття рішень, пов'язану з оптимізацією транспортного маршруту від початкової точки до кінцевої.

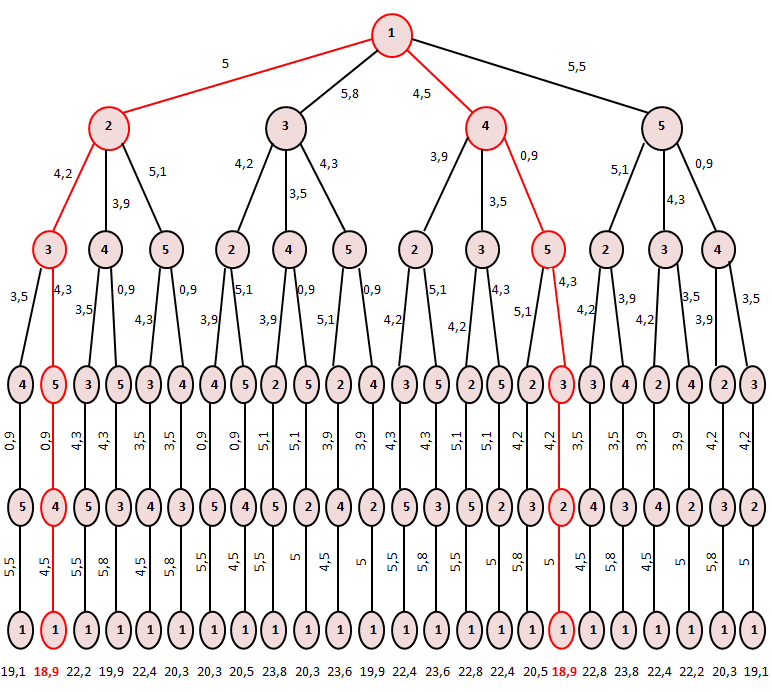
У м. Маріуполі існує одна з найбільших дистриб'юторських компаній – ТОВ «Співдружність», що займається виробництвом борошняних кондитерських виробів. Ця організація постачає продукцію в різні торгові точки. Розробимо маршрут з найменшими витратами зі складу компанії по точкам мережі магазинів «Щирий кум» в Центральному районі міста і повернення транспорту на склад.

Складемо математичну модель даної задачі за допомогою графу (сукупності точок, які називаються вершинами, деякі з яких з'єднані дугами). Торгові точки, які планує об’їздити машина з продукцією та сам склад, позначимо у вигляді кружечків з цифрами – Склад ТОВ «Співдружність» – (1), точки мережі магазинів «Щирий кум» – (2), (3), (4), (5). Вага кожного ребра – відстань між зазначеними місцями. Позначимо їх на ребрах графу у кілометрах (*рис. 2.2.1.*).

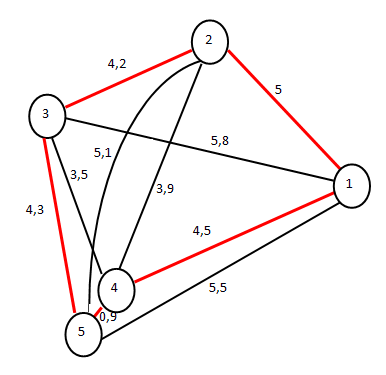
******

*Рис. 2.2.1. Граф до задачі оптимізації маршруту*

Побудуємо математичну модель, де граф має вигляд орієнтованого дерева (*рис. 2.2.2.*).



*Рис. 2.2.2. Граф – орієнтоване дерево*



*Рис. 2.2.3. Оптимальний маршрут*

Отже, маршрутом з найменшими витратами, який має мінімальну довжину 18,9 км: Склад ТОВ «Співдружність» (1), точки мережі магазинів «Щирий кум» (2),(3),(5),(4), Склад ТОВ «Співдружність» (1) (*рис. 2.2.3.*).

# РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАДАЧІ ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ МАРШРУТІВ

# 3.1. Вибір IDE для розробки системи оптимізації транспортних маршрутів

Qt Creator – це крос-платформне інтегроване середовище розробки Qt-додатків, яке надає інструменти проектування та розробки складних програм, на мовах програмування таких, як С, С++, QML [12]. В це середовище включений зручний відладчик, а також різні візуальні засоби розробки інтерфейсу, що використовують QtWidgets і QML. Творці даного програмного рішення ставили собі ціль – створити засіб, що значно спрощує розробку додатків за допомогою фреймворка Qt для різних платформ.

QT Creator дозволяє виконувати вузькоспеціалізовані функції, наприклад, такі як налагодження додатків на QML і відображення даних в відладчику з контейнерів Qt. Середовище розробки підтримує проекти таких засобів, як Qmake, Cmake, Autotools. А для тих проектів, які були створені за допомогою інших програм, була передбачена можливість використання середовища в якості редактора вихідних програмних кодів. До інших зручних функцій Qt Creator відноситься реалізація авто-доповнення, підсвічування коду, а також можливість завдання стилів вирівнювання, постановки дужок і відступів. Загалом, перед нами один з кращих засобів у своїй категорії.

Серед підтримуваних компіляторів: GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW.

Qt дозволяє запускати програмне забезпечення, яке було написане з його допомогою, в більшості сучасних операційних систем шляхом простої компіляції програми для кожної системи без зміни вихідного коду. Включає в себе всі основні класи, які можуть знадобитися при розробці прикладного програмного забезпечення, починаючи від елементів графічного інтерфейсу і закінчуючи класами для роботи з мережею, базами даних і XML. Є повністю об'єктно-орієнтованим, розширюваним і підтримує техніку компонентного програмування.

Використання метаоб'ектного компілятора є особливістю середовища – попередньої системи обробки вихідного коду. Розширення можливостей забезпечується системою плагінів, які можливо розміщувати безпосередньо в панелі візуального редактора. Також існує можливість розширення звичної функціональності віджетів, пов'язаної з розміщенням їх на екрані, відображенням, перемальовуванням при зміні розмірів вікна.

Qt Creator поставляється з редактором коду і Qt Designer для проектування і складання графічних інтерфейсів користувача (GUI) з віджетів Qt.

У програмі передбачена функція збереження результатів для подальшої роздруківки. Визначає вид Головного вікна, розташування і вид віджетів розміщених на головному вікні. Містить інформацію про підключених сигналах і слотах для віджетів.

Інтерфейс програми складається з головного вікна програми, на якому розташовані такі елементи функціональні елементи (віджети):

* Простір введення числових значень (таблиця відстаней, Віджет Таблиця), простір змінюється за допомогою лічильника кількості міст.
* Лічильник міст дозволяє задати розмірність матриці відстаней.
* Кнопка (Віджет кнопка) «Обработать» запускає процес розрахунку - алгоритм Літла, згідно з яким програма здійснює пошук оптимального шляху.
* Вікно виведення текстового результату (Віджет Текстове поле) після виконання обробки надає користувачеві інформацію про оптимальний маршрут.
* Кнопка (Віджет кнопка) «Сохранить результат» – зберігає отриманий результат побудови оптимального шляху в .txt файлі - «Маршрутний лист». Після збереження результатів матриця відстаней скидається.

# 3.2. Оптимізація транспортних маршрутів підприємства «Співдружність»

В попередньому розділі було з’ясовано як працює компанія ТОВ «Співдружність» з точки зору постачання продукції. Тож оптимізуємо транспортні витрати цієї організації за допомогою системи Qt.

У якості вихідних значень для роботи програми використовують попередньо складені матриці відстаней. За допомогою сервісу google maps були оцінені відстані між точками маршрутів та складені матриці відстаней. Адреси та матриці відстаней зображено на таблицях1-2.

*Таблиця 1. Адреси точок об’їзду машини з продукцією.*

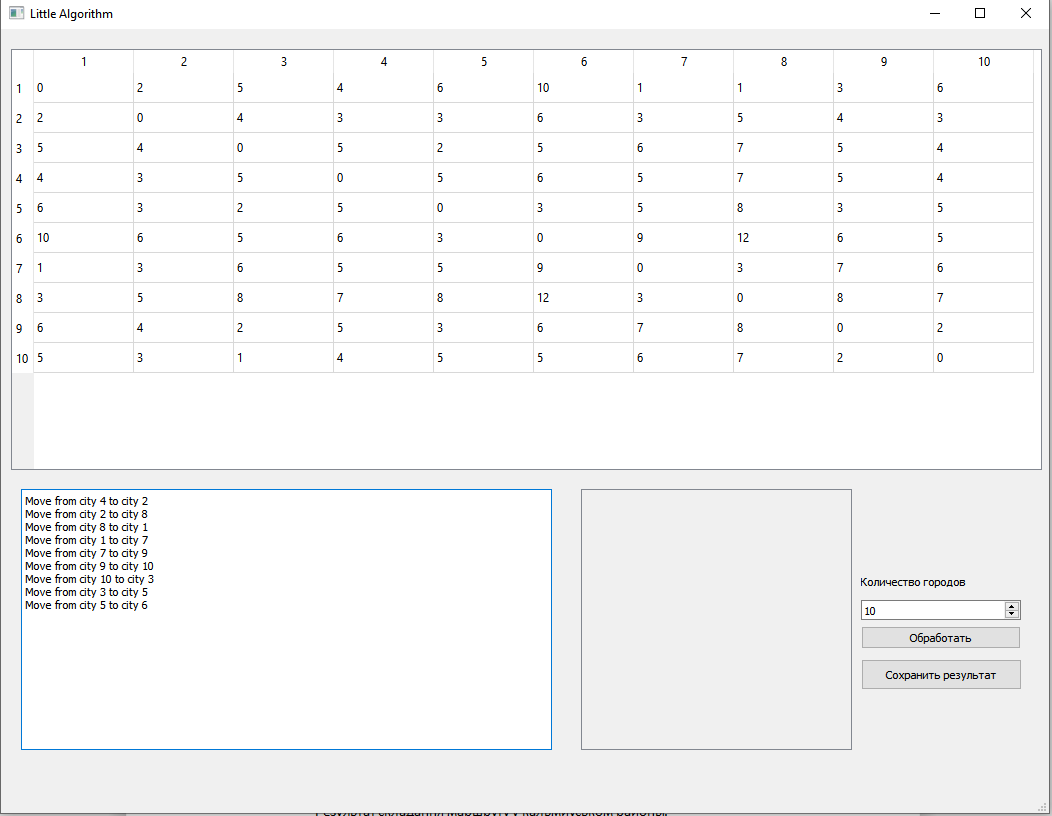
|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва | Адреса | буд. |
| 1 | Щирий кум | вул. Університетська | 6 |
| 2 | Щирий кум | просп. Будівельників | 125 |
| 3 | Щирий кум | просп. Металургів | 211 |
| 4 | Магазин Щирий кум | вул. Гранітна | 33 |
| 5 | Щирий кум | вул. Пилипа Орлика | 122 |
| 6 | Shchyryy Kum | просп. Луніна | 29 |
| 7 | Щирий кум | вул. Митрополитська | 64-63 |
| 8 | Щирий кум | просп. Миру | 110а |
| 9 | Грація | вул. Пилипа Орлика | 88 |
| 10 | Shchyryy Kum | Бул. Шевченка | 295а |

*Таблиця 2. Матриця відстаней точок для таблиці 1.*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | М | 2 | 5 | 4 | 6 | 10 | 1 | 3 | 6 | 5 |
| 2 | 2 | М | 4 | 3 | 3 | 6 | 3 | 5 | 4 | 3 |
| 3 | 5 | 4 | М | 5 | 2 | 5 | 6 | 8 | 2 | 1 |
| 4 | 4 | 3 | 5 | М | 5 | 8 | 5 | 7 | 5 | 4 |
| 5 | 6 | 3 | 2 | 5 | М | 3 | 5 | 8 | 3 | 2 |
| 6 | 10 | 6 | 5 | 6 | 3 | М | 9 | 12 | 6 | 5 |
| 7 | 1 | 3 | 6 | 5 | 5 | 9 | М | 3 | 7 | 6 |
| 8 | 3 | 5 | 8 | 7 | 8 | 12 | 3 | М | 8 | 7 |
| 9 | 6 | 4 | 2 | 5 | 3 | 6 | 7 | 8 | М | 2 |
| 10 | 5 | 3 | 1 | 4 | 5 | 5 | 6 | 7 | 2 | М |

Програма працює наступним чином – користувач повинен вибрати необхідне числове значення лічильника міст, що буде визначати розмірність матриці відстаней. Потім користувач вручну вводить числові значення матриці відстаней. Для побудови оптимального маршруту потрібно натиснути кнопку «Обробка». Натискання цієї кнопки запускає процес роботи алгоритму Літтла для складання оптимального маршруту. Після відпрацювання алгоритму програма видає результат в текстовому полі виведення. Для збереження результату в текстовому файлі необхідно натиснути кнопку «Зберегти результат». Результати моделювання програми зображено на рис. 3.

Розробка системи та її подальше використання на підприємстві має позитивний економічний ефект. Завдяки оптимізації транспортних маршрутів можна не тільки скоротити витрати на пальне та зменшити час виконання проектів, але й підвищити ефективність роботи.

*Рис.3. Результат складання маршруту.*

# ВИСНОВОК

У даній науковій роботі, ми розглянули основні теоретичні аспекти динамічної задачі пошуку оптимального транспортного шляху та дослідили моделі оптимізації маршрутів на основі графового подання ситуацій.

Подано огляд математичних моделей пошуку оптимальних шляхів і аналіз відповідних сервісів, які включають в собі багатофункціональні системи, що можуть автоматично планувати та розраховувати оптимальні маршрути.

Було проаналізовано основні підходи для пошуку найвигідніших шляхів: алгоритм Дейкстри, динамічне програмування та ієрархічні підходи. Проаналізувавши їх, розв’язали за допомогою алгоритму Дейкстри задачу, що полягала знаходження самого вигідного маршруту, що проходить із заданої точки до інших точок та знайшли найкоротший шлях. Отримані результати показали, що запропонований алгоритм забезпечує прийнятну якість результатів. Удосконалено математичну модель оптимізації транспортних маршрутів на основі графового подання для підприємства «Співдружність».

Розглянули середовище проектування та розробки складних програм Qt-додатків – Qt Creator. На цій основі було вирішено завдання, в ході якого була успішно розроблена система оптимізації транспортних витрат. Як результат, ця система може бути використана для інших проектів та задач, які потребують складання найкращих маршрутів.

.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. В. В. Бібліченко, В. П. Кужель. Моделювання технологічних процесів підприємств автомобільного транспорту [Електронний ресурс] // Розділ 1. Моделювання. Основні поняття – Режим доступу: https://web.posibnyky.vntu.edu.ua/fmbt/avto6\_bilichenko\_modelyuvtehproces\_avtotransportu/p2.html.
2. Е. Додонова. [Информатика.](https://sites.google.com/a/gkl-kemerovo.ru/informatics/models) [Електронний ресурс] // Информационные модели. Модели на графах. – 2013. – Режим доступу: https://sites.google.com/a/gkl-kemerovo.ru/informatics/models/modeli-na-grafah.
3. Мурашина логістика [Електронний ресурс] // 2013. – Режим доступу: https://ant-logistics.com/uk/main.html#.
4. EasyWay общественный транспорт [Електронний ресурс] // Режим доступу: https://play.google.com/store/apps/details?id=com.eway&hl=ru.
5. GoogleКарты [Електронний ресурс] // Режим доступу: https://www.google.com/intl/ru/maps/about/.
6. Л.Ф. Гуляницький, А.І. Павленко. Динамічна задача пошуку найкоротшого шляху з додатковими умовами при побудові маршруту авіаперельотів. [Електронний ресурс] // Математичне моделювання в економіці, №2 – 2015. – Режим доступу: http://nbuv.gov.ua/j-pdf/mmve\_2015\_2\_6.pdf.
7. Лабораторна робота №1. Алгоритм Дейкстри. [Електронний ресурс] // ОКМ\_методичні вказівки\_v001. – Режим доступу: https://iq.vntu.edu.ua/fdb/896/%D0%9E%D0%9A%D0%9C\_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BD%D1%96\_%D0%B2%D0%BA%D0%B0%D0%B7%D1%96%D0%B2%D0%BA%D0%B8\_v001.pdf.
8. Тема 8. Динамічне програмування. [Електронний ресурс] // Алгоритми і структури даних. – Конспект лекції. – 2006-2018. – Режим доступу: https://dl.sumdu.edu.ua/textbooks/95351/522257/index.html.
9. А. Павленко. Моделювання і оптимізація маршрутів у транспортних мережах. [Електронний ресурс] // Київ, 2018. – Режим доступу: http://incyb.kiev.ua/wp-content/uploads/2019/04/pavlenko-diser-finalna-1-edited.pdf.
10. Л.Ф. Гуляницький, А.І. Павленко. Моделювання залежних від часу проблем пошуку оптимальних маршрутів: огляд. [Електронний ресурс] // Математичне моделювання в економіці, №1-2 – 2017. – Режим доступу:http://dspace.nbuv.gov.ua/bitstream/handle/123456789/131908/08-Hulianytskyi.pdf?sequence=1.
11. Алгоритм Дейкстры. Поиск оптимальных маршрутов на графе. [Електронний ресурс] // 2011. – Режим доступу: <https://habr.com/ru/post/111361/>.
12. Qt Creator. [Електронний ресурс]. Режим доступу: <https://ru.wikipedia.org/wiki/Qt_Creator>.

# ДОДАТОК А

*Лістинг програми*

**Algorithm.h**

**#pragma once**

#include <string>

#include <vector>

class **Algorithm**

{

public:

char\* name = "Algorithm"; // Название алгоритма

std::vector<std::vector<int>> data; // Массив значений (матрица)

**Algorithm**();

**Algorithm**(std::vector<std::vector<int>>);

**Algorithm**(char\*);

bool **LoadData**(std::vector<std::vector<int>>);

bool **LoadData**(char\*);

virtual void ***Run***(); // Метод для запуска алгоритма

protected:

int **GetStrCount**(std::ifstream&); // Считываем количество строк в файле

int **GetColCount**(std::ifstream&); // Считываем количество столбцов в файле

virtual bool ***validateData***(); // Метод для проверки значений в матрице. Вызывается перед Run()

};

**LittleAlgorithm.h**

**#pragma once**

#include "Algorithm.h"

using namespace std;

class **LittleAlgorithm** : public Algorithm

{

public:

vector<pair<int,int>> result; // Результат. Здесь будет храниться искомый цикл

**LittleAlgorithm**();

**LittleAlgorithm**(vector<vector<int>>);

**LittleAlgorithm**(char\*);

virtual void ***Run***();

private:

enum **check**{Row, Col};

int **getMin**(vector<vector<int>>, int, check); // Поиск минимального элемента столбца/строки

void **matrixProcedure**(vector<vector<int>>); // Метод в котором идет поиск цикла

void **showMatrix**(vector<vector<int>>); // Вывод матрицы

int **getResultSum**(); // Считывание суммы всех выбранных дуг

virtual bool ***validateData***();

};

**mainwindow.h**

**#ifndef MAINWINDOW\_H**

#define MAINWINDOW\_H

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

public:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**myedit.h**

**#ifndef MYEDIT\_H**

#define MYEDIT\_H

#include "LittleAlgorithm.h"

#include "QTextEdit"

class **MyEdit** : public QTextEdit

{

Q\_OBJECT

public:

explicit **MyEdit**(QWidget \*parent = nullptr);

public slots:

void **print**(QString result)

{

this->insertPlainText(result);

}

};

#endif // MYEDIT\_H

**mytable.h**

**#ifndef TABLE\_H**

#define TABLE\_H

#include <QTableWidget>

#include <QObject>

#include <LittleAlgorithm.h>

#include <iostream>

#include <QDebug>

#include "string"

#include "QTextEdit"

using namespace std;

class **table** : public QTableWidget

{

Q\_OBJECT

public:

explicit **table**(QWidget \*parent = nullptr);

signals:

void **Write**(QString result);

public slots:

void **ChangeSize**(int index)

{

setRowCount(index);

setColumnCount(index);

}

vector<pair<int,int>> **Process**()

{

LittleAlgorithm \*alg=new LittleAlgorithm();

vector<vector<int>> \*input=new vector<vector<int>>();

for(int i=0;i<this->rowCount();i++)

{

vector<int> a;

for(int j=0; j<this->rowCount();j++)

{

a.push\_back(stoi((this->item(i,j)->text().toStdString())));

}

input->push\_back(a);

}

alg->LoadData(\*input);

alg->*Run*();

QString a;

for(int i=0;i<this->columnCount();i++)

{

a.insert(a.size(),"Move from city "+QString::number(alg->result[i].first)+" to city "+QString::number(alg->result[i].second)+"\n");

}

cout<<a.toStdString()<<endl;

Write(a);

return alg->result;

}

};

#endif // TABLE\_H

**Algorithm.cpp**

**#include "Algorithm.h"**

#include <iostream>

#include <fstream>

#include <vector>

#include "string.h"

using namespace std;

Algorithm::**Algorithm**() {}

//load functions

Algorithm::**Algorithm**(std::vector<std::vector<int>> Data)

{

LoadData(Data);

}

Algorithm::**Algorithm**(char\* Path)

{

LoadData(Path);

}

int Algorithm::**GetStrCount**(ifstream &file)

{

int count = 0;

while (!file.eof()) {

file.getline(new char[255], 255);

count++;

};

file.seekg(0);

return count;

}

int Algorithm::**GetColCount**(ifstream &file)

{

int count = 0;

char str[255];

file.getline(str, 255);

for (int i = 0; i < strlen(str); i++)

{

if (isdigit(str[i]))

if (i != 0 && !isdigit(str[i - 1]))

count++;

else if (i == 0)

count++;

}

file.seekg(0);

return count;

}

bool Algorithm::**LoadData**(vector<vector<int>> Data)

{

data = Data;

return true;

}

bool Algorithm::**LoadData**(char\* Path)

{

ifstream file(Path, ios\_base::in);

if (!file.is\_open()) throw "Can't open data file";

int rows = GetStrCount(*file*), cols = GetColCount(*file*);

for(int i = 0; i < rows; i++)

{

vector<int> temp;

for(int j = 0; j < cols; j++)

{

int n;

file >> n;

temp.push\_back(n);

}

data.push\_back(vector<int>(temp));

}

file.seekg(0);

return true;

}

//virtual functions

bool Algorithm::***validateData***()

{

return true;

}

void Algorithm::***Run***()

{

system("cls");

if (!*validateData*()) throw "Not valid data.";

std::cout << name << ":" << std::endl;

}

**LittleAlgorithm.cpp**

void LittleAlgorithm::***Run***()

{

name = "Little algorithm";

Algorithm::Run();

matrixProcedure(vector<vector<int>>(data));

}

int LittleAlgorithm::**getMin**(vector<vector<int>> matrix, int sel, check pos)

{

int min = INT32\_MAX;

for (int i = 0; i < matrix[sel].size() - 1; i++)

switch (pos)

{

case LittleAlgorithm::Row:

if (min > matrix[sel][i])

min = matrix[sel][i];

break;

case LittleAlgorithm::Col:

if (min > matrix[i][sel])

min = matrix[i][sel];

break;

}

return min;

}

void LittleAlgorithm::**showMatrix**(vector<vector<int>> temp)

{

std::cout << endl;

std::cout << "\t";

for (int i = 0; i < temp[temp.size() - 1].size() - 1; i++) {

std::cout << temp[temp.size() - 1][i] << "\t";

}

std::cout << endl;

std::cout << "\t";

for (int i = 0; i < temp[0].size(); i++)

for (int j = 0; j<6; j++) std::cout << "\_";

std::cout << endl <<endl;

for (int i = 0; i < temp.size() - 1; i++) {

std::cout << temp[i][temp.size() - 1] << " | " << "\t";

for (int j = 0; j < temp[i].size() - 1; j++)

if(temp[i][j] != INT32\_MAX && j != temp.size() - 1)std::cout << temp[i][j] << "\t";

else std::cout << "inf" << "\t";

std::cout << endl;

}

std::cout << endl << endl;

}

int LittleAlgorithm::**getResultSum**()

{

int sum = 0;

for (int i = 0; i < result.size(); i++)

sum += data[result[i].first - 1][result[i].second - 1];

return sum;

}

bool LittleAlgorithm::***validateData***()

{

//Добавляем в нашу матрицу столбец и строку с нумерацией для отслеживания удаляемых ребер

for (int i = 0; i < data.size(); i++)

for (int j = 0; j < data[i].size(); j++)

if (data[i][j] == 0)

data[i][j] = INT32\_MAX;

vector<vector<int>> temp(data);

for (int i = 0; i < data.size(); i++)

data[i].push\_back(i + 1);

vector<int> numeration;

for (int i = 0; i < data[0].size(); i++)

numeration.push\_back(i + 1);

data.push\_back(numeration);

return true;

}

void LittleAlgorithm::**matrixProcedure**(vector<vector<int>> matrix)

{

//Определяем точку возврата и удаляем необходимое ребро

if (matrix.size() - 1 > 2){

vector<int> vertexes;

for (int i = 0; i < result.size(); i++) {

vertexes.push\_back(result[i].first);

vertexes.push\_back(result[i].second);

}

for (int i = 0; i < vertexes.size(); i++) {

pair<int, int> elem(INT32\_MAX, INT32\_MAX), elem1(INT32\_MAX, INT32\_MAX);

for (int j = 0; j < vertexes.size(); j++) {

if (vertexes[i] != vertexes[j]) {

for (int k = 0; k < matrix[matrix.size() - 1].size() - 1; k++) {

if (vertexes[i] == matrix[k][matrix[k].size() - 1]) elem.first = k;

if (vertexes[j] == matrix[k][matrix[k].size() - 1]) elem1.first = k;

}

for (int k = 0; k < matrix.size() - 1; k++) {

if (vertexes[i] == matrix[matrix.size() - 1][k]) elem.second = k;

if (vertexes[j] == matrix[matrix.size() - 1][k]) elem1.second = k;

}

}

}

for (int i = 0; i < matrix.size() - 1; i++)

for (int j = 0; j<matrix.size() - 1; j++)

if (i == elem1.first && j == elem1.second)

matrix[elem1.first][elem1.second] = INT32\_MAX;

for (int i = 0; i < matrix.size() - 1; i++)

for (int j = 0; j < matrix.size() - 1; j++)

if (i == elem.first && j == elem.second)

matrix[elem.first][elem.second] = INT32\_MAX;

}

}

//Вычитаем из каждой строки минимальное значение

for (int i = 0; i < matrix.size() - 1; i++) {

int min = 0;

if ((min = getMin(matrix, i, check::Row)) == INT32\_MAX) {

showMatrix(matrix);

std::cout << endl << "Bad road" << endl;

return;

}

if ((min = getMin(matrix, i, check::Row)) != 0)

for (int j = 0; j < matrix[i].size() - 1; j++)

if(matrix[i][j] != INT32\_MAX) matrix[i][j] -= min;

}

//Вычитаем из каждого столбца минимальное значение

for (int i = 0; i < matrix[matrix.size() - 1].size() - 1; i++) {

int min = 0;

if ((min = getMin(matrix, i, check::Col)) == INT32\_MAX) {

showMatrix(matrix);

std::cout << endl << "Bad road" << endl;

return;

}

if ((min = getMin(matrix, i, check::Col)) != 0)

for (int j = 0; j < matrix.size() - 1; j++)

if (matrix[j][i] != INT32\_MAX) matrix[j][i] -= min;

}

//Находим максимально оцененный ноль

int Max = 0;

for (int i = 0; i < matrix.size() - 1; i++)

for (int j = 0; j < matrix[i].size() - 1; j++)

if (matrix[i][j] == 0) {

matrix[i][j] = INT32\_MAX;

int max = (getMin(matrix, i, check::Row) == INT32\_MAX || getMin(matrix, j, check::Col) == INT32\_MAX)? INT32\_MAX: getMin(matrix, i, check::Row) + getMin(matrix, j, check::Col);

if (max > Max) Max = max;

matrix[i][j] = 0;

}

//Находим все нули максимальная оценка которых равна Max

vector<pair<int, int>> Maxs;

for (int i = 0; i < matrix.size() - 1; i++)

for (int j = 0; j < matrix[i].size() - 1; j++)

if (matrix[i][j] == 0) {

matrix[i][j] = INT32\_MAX;

int max = (getMin(matrix, i, check::Row) == INT32\_MAX || getMin(matrix, j, check::Col) == INT32\_MAX) ? INT32\_MAX : getMin(matrix, i, check::Row) + getMin(matrix, j, check::Col);

if (max == Max) Maxs.push\_back(pair<int, int>(*matrix[i][matrix.size()* *-* 1*]*, *matrix[matrix.size()* *-* 1*][j]*));

matrix[i][j] = 0;

}

//Вывод координат выбранных нулей

std::cout << "Maxs - ";

for (int i = 0; i < Maxs.size(); i++)

std::cout << Maxs[i].first << " " << Maxs[i].second << "\t";

std::cout << endl;

//Вывод матрицы

showMatrix(matrix);

std::cout << endl;

//Завершаем выполнение данной ветви если нету нулей

if (Maxs.size() == 0) {

std::cout << "Bad road." << endl;

return;

}

for (int i = 0; i < Maxs.size(); i++) {

//Добавляем вершину в массив с результатом

result.push\_back(Maxs[i]);

//Если размер матрицы порядка 1, выводим результат и завершаем текущию ветвь

if (matrix.size() - 1 == 1) {

for (int i = 0; i < result.size(); i++)

std::cout << "(" << result[i].first << ", " << result[i].second << ")\t";

std::cout << endl;

std::cout << "Result: " << getResultSum() << endl;

result.pop\_back();

return;

}

//Создаем копию текущей матрицы и удаляем из нее строку и столбец выбранного нуля

vector<vector<int>> temp(matrix);

pair<int, int> elem(INT32\_MAX, INT32\_MAX), elem1(INT32\_MAX, INT32\_MAX);

for (int j = 0; j < temp[temp.size() - 1].size() - 1; j++) {

if (Maxs[i].first == temp[j][temp[j].size() - 1]) elem.first = j;

if (Maxs[i].second == temp[j][temp[j].size() - 1]) elem1.first = j;

}

for (int j = 0; j < temp.size() - 1; j++) {

if (Maxs[i].second == temp[temp.size() - 1][j]) elem.second = j;

if (Maxs[i].first == temp[temp.size() - 1][j]) elem1.second = j;

}

for(int i = 0; i < temp.size() - 1; i++)

for(int j = 0;j<temp.size() - 1; j++)

if(i == elem1.first && j == elem1.second)

temp[elem1.first][elem1.second] = INT32\_MAX;

for (int j = 0; j < temp[temp.size() - 1].size(); j++)

temp[j].erase(temp[j].begin() + elem.second);

temp.erase(temp.begin() + elem.first);

//Вызываем рекурсивно эту же функцию для уже новой матрицы

matrixProcedure(temp);

//Удаляем последние значение из массива с результатом

result.pop\_back();

}

}

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QTableWidget>

#include <QPushButton>

#include <QComboBox>

#include <QObject>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

MainWindow w;

w.setWindowTitle("Little Algorithm");

w.show();

return a.exec();

}

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <table.h>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~***MainWindow***()

{

delete ui;

}

**myedit.cpp**

#include "myedit.h"

MyEdit::**MyEdit**(QWidget \*parent):QTextEdit(parent){;}

**mytable.cpp**

#include "table.h"

table::**table**(QWidget \*parent):QTableWidget(parent){;}