МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ ТА СИСТЕМНОГО АНАЛІЗУ

До захисту допустити:
Зав. кафедри Шабельник Т.В.
«____ » _____ 20 __р.

Кваліфікаційна робота
за освітнім ступенем «Магістр» на тему:
«Оптимізація транспортних витрат для підприємства ІТ-сфери»

Студентки економіко-правового факультету
спеціальності 124 «Системний аналіз»
освітнього ступеня «Магістр»
Піскорської Олени Вячеславівни
Науковий керівник:
Альохін О.Б.
д.е.н., професор кафедри математичних методів та системного аналізу
Рецензент:
Балалаєва О.Ю, к. т. н., доцент кафедри інформатики, декан факультету інформаційних технологій ДВНЗ “Приазовський державний технічний університет”

Кваліфікаційна робота захищена
з оцінкою _____
Секретар ЕК _____
« ___»_____20 __ р.

Маріуполь – 2020р

МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО – ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ ТА СИСТЕМНОГО АНАЛІЗА

Освітній ступінь «Магістр»
Шифр та назва спеціальності 124 «Системний аналіз»

ЗАТВЕРДЖУЮ
Завідувач кафедри математичних методів та системного аналізу
_____ Т. В. Шабельник

«____» _____202_ р.

ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

_____Піскорської Олени Вячеславівни_ _____
(прізвище, ім'я, по батькові)

1. Тема роботи: Оптимізація транспортних витрат для підприємства ІТ-сфери
керівник роботи кандидат технічних наук, доцент, Альохін О.Б_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом Маріупольського державного університету від «__» _____ 20_____ року №_____
2.Строк подання студентом роботи_____
3. Вихідні дані до роботи (мета, об'єкт, предмет)

Мета роботи полягає в проектуванні системи оптимізації транспортних витрат ІТ-підприємства.

Об'єктом дослідження є процеси транспортування вантажів. підприємства ІТ-сфери.

Предметом дослідження є методи оптимізації транспортних витрат підприємства ІТ-сфери.

4. Зміст роботи (перелік питань, які потрібно розробити)
ВСТУП

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ПРОЦЕСІВ ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ ОПЕРАЦІЙ

1.1 Використання комбінаторної оптимізації для скорочення транспортних витрат

1.2 Постановка завдання комівояжера та класифікація його різновидів

ВИСНОВКИ ДО РОЗДІЛУ 1

РОЗДІЛ 2 АНАЛІЗ МОДЕЛЕЙ ТА МЕТОДІВ ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ ОПЕРАЦІЙ

2.1 Аналіз методів вирішення задачі комівояжера

2.2 Вибір методу оптимізації транспортних витрат

ВИСНОВКИ ДО РОЗДІЛУ 2.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ МЕТОДІВ ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ ОПЕРАЦІЙ

3.1 Характеристика підприємства «Інтегровані системи безпеки»

3.2 Постановка задачі оптимізації транспортних витрат підприємства «Інтегровані системи безпеки»

3.3 Математична та алгоритмічні моделі методу гілок та меж оптимізації транспортних операцій

3.4 Вибір IDE для розробки системи оптимізації транспортних витрат

3.5 Інформаційна модель системи оптимізації транспортних витрат

3.6 Програмна модель системи оптимізації транспортних витрат

3.7 Інтерфейс системи оптимізації транспортних витрат

3.8 Результати моделювання

ВИСНОВКИ ДО РОЗДІЛУ 3

ВИСНОВКИ

Літературні джерела

Додаток А

5. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1. Теоретичні основи процесів оптимізації транспортних операцій	**д.е.н., професор Альохін О.Б.**		
Розділ 2. Аналіз моделей та методів оптимізації	**д.е.н., професор Альохін О.Б.**		

транспортних витрат			
Розділ 3. Реалізація методів оптимізації транспортних витрат	**д.е.н., професор Альохін О.Б.**		

6. Дата видачі завдання 11березня 2020р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів роботи	Примітка
1	Аналіз літературних джерел з теми: «Оптимізація транспортних витрат підприємства IT-сфери»	30.09-02.10.2020	
2	Робота та формування матеріалів параграфу 1.1 Використання комбінаторної оптимізації для скорочення транспортних витрат	03.10 – 05.10-2020	
3	Робота та формування матеріалів параграфу 1.2 Постановка завдання комівояжера та класифікація його різновидів	06.10 – 08.10.2020	
4	Робота та формування матеріалів параграфу 2.1 Аналіз методів вирішення задачі комівояжера	09.10 – 11.10.2020	
5	Робота та формування матеріалів параграфу 2.2 Вибір методу оптимізації транспортних витрат	12.10 – 14.10.2020	
6	Робота та формування матеріалів параграфу 3.1 Характеристика підприємства «Інтегровані системи безпеки»	15.10 – 17.10.2020	
7	Робота та формування матеріалів параграфу 3.2 Постановка задачі оптимізації транспортних витрат підприємства «Інтегровані системи безпеки»	18.10 – 20.10.2020	
8	Робота та формування матеріалів параграфу 3.3 Математична та алгоритмічні моделі методу гілок та меж оптимізації транспортних операцій	21.10 – 23.10.2020	
9	Робота та формування матеріалів параграфу 3.4 Вибір IDE для розробки	30.09-02.10.2020	

	системи оптимізації транспортних витрат		
10	Робота та формування матеріалів параграфів 3.5-3.8. Інформаційна модель системи оптимізації транспортних витрат, Програмна модель оптимізації системи транспортних витрат, Інтерфейс системи оптимізації транспортних витрат, Результати моделювання	23.10 – 27.10.2020	
11	Формування висновків кваліфікаційної роботи	30.10 – 31.10.2020	
12	Оформлення кваліфікаційної роботи	01.11 – 05.11.2020	
13	Підготовка доповіді та презентаційних матеріалів	05.11 – 06.11.2020	

Студент _____ Піскорська О.В
(підпис) (прізвище та ініціали)

Науковий керівник роботи _____ Альохін О.Б.
(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

Актуальність теми. Характерна ознака будь-якого сучасного мегаполісу - високорозвинений ринок товарів і послуг. Перед компаніями, які пропонують такі компоненти споживчого ринку своїм клієнтам, стоїть завдання своєчасно та ефективно організувати свою діяльність та оптимізувати її сфери.

Оптимізація автотранспортних витрат одна із найважливіших проблем підприємств, які пропонують послуги, що будь-якою мірою пов’язані із логістичними процесами – діяльністю по плану фізичного розподілу при виготовленні продукції, переміщенні матеріалів, запасних частин, сировини та готової продукції, завантаженням та розвантаженням, транспортно-складських роботах, тощо. Важливо організувати та оптимізувати робочий процес автотранспортних засобів підприємства таким чином, щоб задовольнити потреби клієнтів, максимізувати свій прибуток та максимально скоротити витрати, виконувати замовлення вчасно та якісно. Виходячи із цього, тема магістерської роботи актуальна.

У зв'язку з важливістю вирішення проблеми оптимізації транспортних витрат підприємства виникає велика кількість завдань, які пов’язані із розв’язками економічних цільових функцій. До класу таких завдань, які можуть застосовуватися для оптимізації ведення господарської діяльності та потреб підприємств, відносяться задачі комбінаторної оптимізації: завдання вибору маршруту, задача комівояжера, задача про вісім ферзів — завдання задоволення обмежень, задача про рюкзак, завдання розкрою, задача про призначення, задача про призначення цілей.

Одне з найвідоміших і важливих завдань транспортної логістики і комбінаторної оптимізації - завдання комівояжера або завдання про мандрівного торговця. Суть завдання зводиться до пошуку оптимального шляху, що проходить через проміжні пункти по одному разу і повертається у вихідну точку. Наприклад, знаходження найбільш вигідного маршруту, що

дозволяє комівояжеру відвідати зі своїм товаром певні міста по одному разу і повернутися назад.

Об'єкт дослідеження. Об'єктом дослідження є процеси транспортування вантажів. підприємства ІТ-сфери.

Предмет дослідження. Предметом дослідження є методи оптимізації транспортних витрат підприємства ІТ-сфери.

Мета роботи і завдання дослідження. Мета дослідження полягає в проектуванні системи оптимізації транспортних витрат ІТ-підприємства.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- проаналізувати існуючі підходи та методи комбінаторної оптимізації;
- сформулювати та класифікувати завдання комівояжера;
- проаналізувати існуючі методи та алгоритми вирішення проблеми комівояжера;
- проаналізувати роботу ІТ-підприємства «Інтегровані системи безпеки»;
- побудувати систему оптимізації автотранспортних витрат ІТ-підприємства «Інтегровані системи безпеки».

Методи дослідження. Теоретичну і методологічну основу кваліфікаційної роботи складають наукові методи комбінаторної оптимізації та системного аналізу, методи теорії прийняття рішень, емпіричні методи.

Інформаційну базу дослідження складають наукові праці вітчизняних і зарубіжних вчених з питань оптимізації транспортних витрат підприємства, нормативно-правова база діяльності підприємств в Україні, звітність підприємств.

Дослідженням питань пошуку рішень та інтерпретацій задачі комівояжера, що у подальшому мала вплив на розвиток транспортних операцій та оптимізацію транспортних витрат займались вітчизняні та зарубіжні науковці: У. Гамільтон, К. Менгер, В. Левітін, Х. Вітні, Д. Данциг, Д. Фалкерсон, С. Джонсон.

Наукова новизна.

Отримав подальший розвиток метод гілок та меж для оптимізації задачі транспортування вантажів.

Дані положення є оригінальними і виносяться на захист в представленій роботі.

Практичне значення.

Розроблена система оптимізації транспортних витрат підприємства ІТ-сфери «Інтегровані системи безпеки». Система є універсальною та придатною для використання будь-яким підприємством ІТ-сфери України для оптимізації автотранспортних витрат.

Апробація результатів дослідження. Основні результати кваліфікаційної роботи доповідалися на Декаді студентської науки та обговорювалися у збірнику тез доповідей студентів Маріупольського Державного Університету економіко-правового факультету «Дебют» 2020, секція Математичні методи, інформаційні технології в освіті, науці, економіці і виробництві - доповідь «Оптимізація транспортних операцій методом гілок та меж».

Структура та обсяг кваліфікаційної роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків і списку використаних джерел. Загальний обсяг роботи становить 83 сторінки, містить 14 таблиць, 26 рисунків. Список використаних літературних джерел налічує 102 найменуваня.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПРОЦЕСІВ ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ ОПЕРАЦІЙ

1.1 Використання комбінаторної оптимізації для скорочення транспортних витрат

Комбінаторна оптимізація - область теорії оптимізації в прикладній математиці, що пов'язана з дослідженням операцій, теорією алгоритмів і теорією обчислювальної складності.

Комбінаторна оптимізація полягає в пошуку оптимального об'єкта в кінцевій множині об'єктів.

Комбінаторна оптимізація часто зводиться до визначення ефективного розподілу ресурсів, що використовуються для пошуку оптимального рішення.

Комбінаторна оптимізація включає в себе завдання оптимізації, в яких множина допустимих рішень дискретна або може бути зведена до дискретної множини.

До окремих завдань комбінаторної оптимізації відносять завдання вибору маршруту, завдання комівояжера, завдання задоволення обмежень, завдання про рюкзак, завдання розкрою, завдання про призначення, та інші.

Завдання комбінаторної оптимізації можна розглядати як пошук кращого елемента в деякій дискретній множині, для вирішення завдань комбінаторної оптимізації можуть бути використані майже будь-які алгоритми пошуку.

Розвиток методів вирішення задач комбінаторної оптимізації обумовлено теоретичної і практичної важливістю цих завдань, що зустрічаються в різних областях науки і техніки. Комбінаторна оптимізація на практиці може бути використана не тільки для визначення оптимальних шляхів руху автотранспорту при виконанні доставки, але і для складання оптимальних транспортних мереж, мереж аерофлоту, для визначення правильних атрибутів перед тестуванням концепцій.

Сучасний розвиток методів йде по шляху створення нових чисельних методів для вирішення як загальних завдань комбінаторної оптимізації, так і завдань приватного виду.

Основна ідея методів вирішення комбінаторних завдань полягає у використанні скінченної множини допустимих рішень і заміні повного їх перебору скороченим або неявним.

Головну роль в скороченні перебору грають оцінка і відкидання неперспективних підмножин, які не містять оптимальних рішень. Відкидання таких підмножин дозволяє замінити повний перебір частковим і, тим самим, зменшити витрати обчислювальних ресурсів.

Нехай існує множина із n елементів. На цій множині задається безліч комбінацій $P = \{p_1, p_2, \ldots, p_n\}$, де під комбінаціями розуміються перестановки, поєднання та підстановки властиві кожній конкретній задачі. На множині P задається деяка функція f. У оптимізаційно-комбінаторної задачі шукається екстремум функції f і відшукуються ті елементи множини P, на яких функція f екстремальна.

У загальному вигляді завдання можна сформулювати так: із заданої множини предметів з властивостями «вартість» і «вага» потрібно відібрати підмножину з максимальною повною вартістю, дотримуючись при цьому обмеження на сумарну вагу.

При вирішенні оптимізаційно-комбінаторних задач потрібно вміти перебирати множину значень функції f, вміти порівнювати та обчислювати ці значення.

Комбінаторні методи можна розрізнити за способом розбиття і способом оцінювання, ці способи звичайно пов'язані із специфікою розв'язуваних класів задач. Правила, відповідно до яких проводиться відсів підмножин називаються правилами відсіву. За принципом роботи комбінаторні методи можна поділити на локальну оптимізацію, випадковий пошук та методи розгалуження.

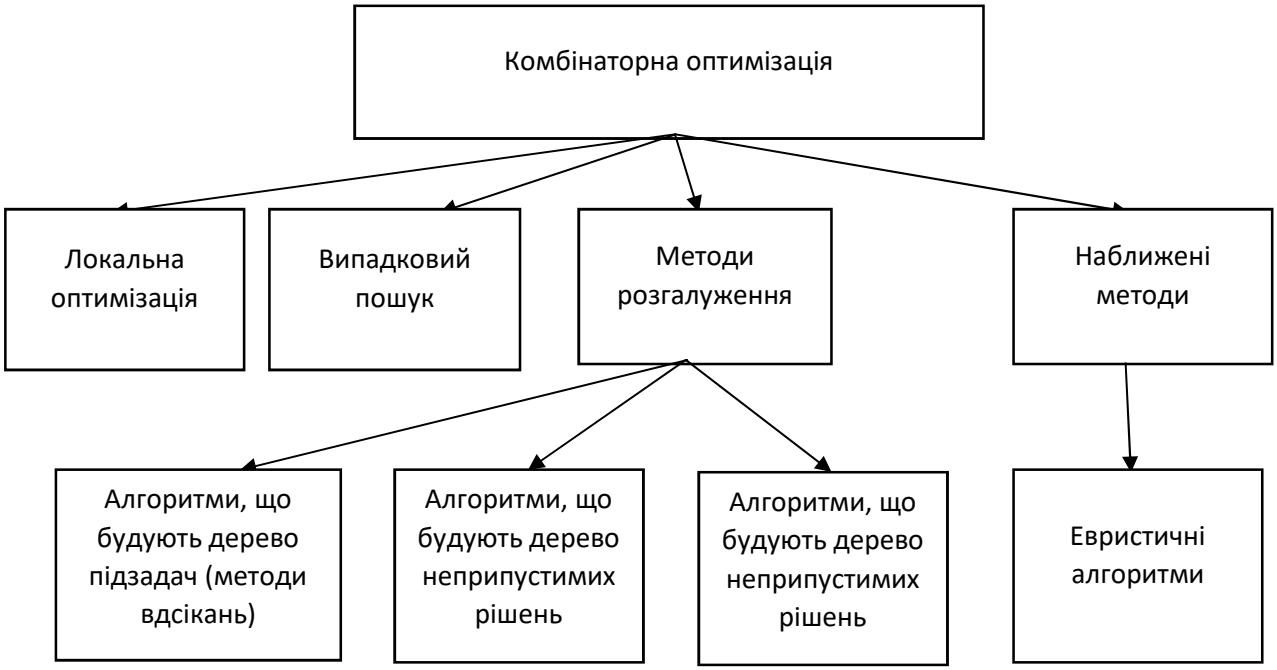


Рисунок 1.1 Комбінаторна оптимізація та її методи

При локальній оптимізації для кожної комбінації $p_i \in P$ визначається Q_i – множина комбінацій, сусідніх з p_i. Вихідною операцією є випадковий вибір початкової комбінації. Потім на множині Q_i знаходять локальний екстремум. Цей процес повторюється багато разів, і серед отриманих локальних екстремумів вибирається найменший - саме він приймається за наближене значення.

При випадковому пошуку вибір всіх комбінацій відбувається випадково відповідно до деякого закону розподілу. Значення цільової функції на цих комбінаціях обчислюються, потім серед них вибирається та комбінація, яка дає екстремальне значення цільової функції.

До методів розгалуження відносять метод гілок та меж. Спершу цей метод був запропонований для вирішення завдань цілочисельного лінійного програмування. А надалі цей метод почали застосовувати до більш загальних класів комбінаторних оптимізаційних задач. За способом розгалуження усі алгоритми гілок і меж можна розділити на наступні групи:

1) Алгоритми, що будують дерево підзадач вихідного завдання. У цьому методі будується дерево завдань лінійного програмування,

домагаючись поступового задоволення умов цілочисельності - варіант методу відсікань.

2) Алгоритми, що будують дерево неприпустимих рішень. Цей метод застосовується для розв'язання задач лінійного програмування з булевими змінними.
3) Алгоритми, що будують дерево допустимих рішень (метод вирішення завдань про комівояжера).

Розглядаючі комбінаторну оптимізації та її завдання потрібно розглянути і наближені методи для вирішення деяких завдань оптимізації.

Наближені методи вирішення задач дозволяють знаходити наближені рішення, так як знаходження точного рішення може потребувати значних обчислювальних ресурсів.

Сучасні наближені методи містять в собі елементи різних методів і є комбінованими. У наближених методах рішення задачі проводять наступним чином - будують початкове рішення, а надалі поліпшують початкове рішення. При цьому на етапі побудови початкового рішення широко використовуються евристичні алгоритми. Такі алгоритми, що засновані на правдоподібних, але не обґрунтованих строго припущеннях про властивості оптимального рішення задачі.

У якості приклада евристичного алгоритму можна навести алгоритм вирішення задачі комівояжера, в якому на кожному кроці реалізується перехід в найближчу з решти точок. Алгоритми такого типу називають «жадібні».

На кожному кроці ці алгоритми вирішують локальну задачу оптимізації, але отримане рішення може бути як завгодно далеким від оптимуму.

Наближені методи базуються на введенні додаткових умов припинення процесу оптимізації та відмові від пошуку точного рішення. Підставою для використання наближених методів можуть бути наступні ознаки:

- якщо відбувається експонентне зростання обсягу обчислень з ростом розмірності завдань.

- коли для точного розв'язання задач великої розмірності не вистачає потужностей обчислювальної машини;

- якщо в практичній задачі вихідну задачу вдається сформулювати лише приблизно;

- коли вихідні дані в прикладній задачі і параметри моделі є наближеними і пошук точного рішення не виправданий.

Від наближених методів, що гарантують пошук рішення із заданою точністю слід відрізняти евристичні методи, які не гарантують точність одержаних рішень.

1.2 Постановка завдання комівояжера та класифікація його різновидів

Розглядаючи завдання комівояжера не можна не згадати про вчених та історію її розвитку.

Одним з перших, хто сформулював ранній варіант цього завдання, був видатний Ірландський математик, фізик і механік – Уільям Роуен Гамільтон. Основою задачі комівояжера можно вважати гру «Ікосіан», мета якої полягала в проходженні всіх вершин дванадцятигранника по одному разу і тільки по його ребрах, з подальшим поверненням у відправну точку, яку вчений запропонував у 1857 році. Іншими словами потрібно було знайти так званий гамільтонів цикл на графі з 20 вузлами[25].

А вже у 1930 році австрійсько-американський математик Карл Менгер сформулював завдання комівояжера як «завдання знайти найкоротший шлях між кінцевою безліччю місць, відстань між якими відомо». Сучасна назву «Завдання комівояжера» отримала від американського математика Хасслера Вітні.

У 1954 році Джордж Данциг, Делберт Рей Фалкерсон і Селмер Джонсон сформолювали цю проблему в якості завдання дискретної оптимізації. Вчені застосували метод відсікань, а саме алгоритм Гоморі для вирішення приватної задачі комівояжера і обґрунтували оптимальність знайденого маршруту.

Надалі дослідження Завдання комівояжера продовжили не тільку у теоретичному, але і у практичному вигляді - вивчалися можливості прикладного застосування у різних областях науки. У 1972 році американський вчений у сфері інформатики Річард Меннінг Карп довів NP-повноту завдання пошуку гамільтонових шляхів, з чого слідувала NP-складність задачі комівояжера [25].

У 1963 році групою авторів у складі Дж. Літла, К. Мурті, Д. Суіні та К. Керол був запропонований метод гілок та меж для вирішення завдання комівояжера[1].

Сучасні поширені методи дискретної оптимізації, такі як метод відсікань, гілок і меж і різні варіанти евристичних алгоритмів, були розроблені саме на прикладі задачі комівояжера.

У 1976 році Нікос Крістофідес винайшов алгоритм, який ефективно знаходить приблизні рішення завдання комівояжера. Однак згодом ні у кого не вийшло поліпшити цей алгоритм. В алгоритмі Крістофідеса все починається з остовного дерева, що з'єднує міста без замкнутих петель. Щоб побудувати його потрібно почати з пошуку найкоротшого шляху між двома містами. Для кожної наступної гілки потрібно знайти найкоротший шлях між новим містом і одним з двох попередніх [90].

Результуюче дерево дозволяє пройти маршрут через кожне місто і повернутися назад, але зазвичай довжина такого шляху далека від найкоротшої. Проте, отриманий шлях в гіршому випадку не перевищує найкоротший більш ніж на 50%.

У 2010 році Амін Сабері зі Стенфордського університету, аспіранти Араш Асадпур, Шайан Гаран і Александер Мадрі, а також Майкл Гоманс показали новий алгоритм, який починається з підрахунку точного дрібного рішення Завдання комівояжера, а потім округлює це рішення до остовного дерева. Нарешті, алгоритм включає це дерево в мережу Крістофідеса. Таким чином, дослідники змогли покращити алгоритм Крістофідеса на невелику частку для широкого підкласу «графічних» завдань комівояжера [90].

Поки поліпшення алгоритму оцінюються в частках відсотка, але, як показує практика, це може бути черговим проривом у вирішенні Завдання комівояжера. Після розробки алгоритму Сабері та інших результат рішення вже покращився з 50 до 40%.

Завдання орієнтування в просторі - в першу чергу це завдання комівояжера, розглянемо та сформулюємо її.

Існує *n* міст, відстані між якими відомі. Комівояжер повинен пройти всі *n* міст по одному разу, повернувшись в той місто, з якого почав. Потрібно знайти такий маршрут руху, при якому сумарна пройдена відстань буде мінімальною [92].

Таким чином мета задачі мета полягає в знаходженні найвигіднішого маршруту, що проходить через всі задані точки по одному разу, з подальшим поверненням у вихідну точку. Умови завдання повинні містити критерій вигідності маршруту - чи повинен він бути максимально коротким, швидким, дешевим, а також вихідні дані у вигляді матриці витрат (відстані, вартості, часу) при переміщенні між розглянутими пунктами.

Особливості задачі в тому, що вона досить просто формулюється і знайти хороші рішення для неї також відносно просто, але разом з тим пошук дійсно оптимального маршруту для великого набору даних - непростий і ресурсномісткий процес. Для вирішення задачі комівояжера її треба представити як математичну модель. При цьому вихідні умови можна записати у форматі матриці-таблиці, де рядкам відповідають міста відправлення, стовпцям-міста прибуття, а в осередках вказуються відстані між ними, або у вигляді графа-схеми, що складається з вершин, які символізують міста, і з'єднують їх ребер, довжина яких відповідає відстані між містами.

Один із способів розв’язку задачі комівояжера – пошук рішення серед гамільтонових циклів, тобто циклів, у яких не відбувається повторного відвідування пунктів.

Мережу доріг для задачі подають як граф $G = (V, E)$,

де V – вершина графа, E – ребра графа;

Вага ребра $c_{ij} > 0$ еквівалент часу проїзду між суміжними вершинами графа.

Для зручності програмної реалізації граф представляють у вигляді матриці, розмірність якої відповідає кількості вершин у графі. Сенс значення елемента матриці d_{ij} ідентичний сенсу ваги ребра c_{ij} у графі, він визначає час проїзду між пунктами i та j. При $i = j$, $d_{ij} = M$. Елемент M символізує нескінченність та забороняє перехід « у себе»[93].

Також задача комівояжера може бути представлена формулою, яка також представляє собою цільову функцію[93]:

$$Z = \sum_1^n = 1 \sum_j^n = 1 \ c_{ij} x_{ij \to min,} , \qquad (1.1)$$

де n – кількість пунктів на карті;

c_{ij} – матриця вартості між пунктами;

x_{ij} – матриця переходів з компонентами;

x_{ij} =1, якщо маршрут включає переїзд із точки і в точку j;

x_{ij} = 0, в іншому випадку.

Постановку задачі можна сформулювати таким чином: знайти мінімум функції Z при виконанні обмежень та невід'ємності значень матриці X.

Система обмежень наступна:

$$\begin{cases} \sum_j^n x_{ij} , i = 1 \dots n \\ \sum_i^n x_{ij} , j = 1 \dots n \\ u_i - u_j + nx \ \leq n - 1, i, j = 1 \dots n, i \neq j \end{cases} , \quad (1.2)$$

де n – кількість пунктів на карті;

x_{ij} - матриця переходів із компонентами;

ui, uj – довільні цілі невід'ємні числа.

При цьому обмеження виражають наступні умови:

1) комівояжер виїжджає із кожного пункту один раз;

2) комівояжер вїжджає в кожен пункт один раз;

3) маршрут не буде містити у собі замкнуті маршрути (умова незамкнутості), крім одного, що включає усі пункти, а також те, що маршрут не повинен містити петель.

Для реалізації наведених рівностей достатньо додоти штрафні функції:

$$c_{kp} > c_{ijmax} \text{ , (1.3)}$$

де $k=p$;

c_{ijmax} – максимальне значення для усіх c_{ij}

Класифікацію видів завдання за симетричністю ребер комівояжера наведено у таблиці 1.1

Табл. 1.1

Класифікація видів завдання комівояжера за ознаками симетричності ребер

Критерій	Характеристика
Симетричність ребер	Задача комівояжера симетрична, коли всі пари ребер, що з'єднують одні і ті ж вершини, мають однакову довжину, тобто граф, що представляє вихідні дані завдання, є неорієнтованим. Іншими словами, довжина прямого шляху від міста А до міста В і довжина зворотного шляху від міста В до міста А, однакові. Те ж саме справедливо і щодо інших пар міст.
Асиметрічність ребер	Задача комівояжера асиметрична, коли довжина пар ребер, що з'єднують одні й ті ж міста, може відрізнятися (орієнтований граф). Для цього типу задачі комівояжера прямий шлях, наприклад, з міста А в місто В може бути коротше або довше зворотного шляху з міста В в місто А. таке може бути не тільки в теорії, але і в реальності - в разі доріг з одностороннім рухом. Крім того, в загальному випадку пошуку найкоротшого шляху охоплює набір пунктів, які потрібно відвідати по одному разу, говорять про узагальнений випадок задачі комівояжера.

Класифікацію видів завдання комівояжера по замкнутості маршруту наведено у таблиці 1.2

Табл. 1.2

Класифікація видів завдання комівояжера по замкнутості маршруту

Критерій	Характеристика
Замкнутість маршруту	Задача замкнута, при знаходженні найкоротшого шляху, що проходить через усі вершини по одному разу з подальшим поверненням в точку старту.
Незамкнутість маршруту	Задача незамкнута, при знаходженні найкоротшого шляху, що проходить через всі вершини по одному разу і без обов'язкового повернення у вихідну точку.

Можна запропонувати й інші класифікації задачі комівояжера наприклад, метрична і неметрична.

Симетричну задачу комівояжера називають метричною [95], якщо щодо довжин ребер виконується нерівність трикутника. В таких завданнях обхідні шляхи довше прямих, тобто ребро від вершини i до вершини j ніколи не бувають довшим шляху через проміжну вершину k: $c_{ij} \leq c_{ik} + c_{kj}$

Якщо в умовах задачі дозволяється відвідувати міста кілька разів, то симетричну задачу можна звести до метричної. Для цього завдання розглядають на так званому графі відстаней. Цей граф має таку ж безліч вершин, як і вихідний, і є повністю зв'язковим. Довжина ребер c_{ij} між вершинами i та j на графі відстаней відповідає довжині найкоротшої відстані між вершинами i та j у вихідному графі. Для визначених таким чином довжин c_{ij} виконується нерівність трикутника, і кожному маршруту на графі відстаней

завжди відповідає маршрут з можливими повтореннями вершин у вихідному графі.

Неметричні завдання комівояжера можуть виникати на разі мінімізації тривалості перебування при наявності вибору транспортних засобів в різних напрямках. В такому випадку пряме повідомлення автомобілем може бути довше обхідного шлях літаком.

ВИСНОВКИ ДО РОЗДІЛУ 1

У сучасному капіталістичному світі час і гроші є ключовими ресурсами. Виходячи з цього оптімазація автотранспортних маршрутів перевезення товарів і надання послуг є ключовим моментом в скороченні фінансових витрат і зменшення витрат часових ресурсів.

Оптимальний автотранспортний маршрут перевезень товарів є найкоротшим маршрутом при використанні якого підприємство здатне скоротити час виконання завдання і значно скоротити витрати на паливо для автотранспортних засобів. У загальному випадку знаходження оптимального маршруту можна виразити за допомогою математичних методів і задач комбінаторної оптимізації.

Комбінаторна оптимізація пропонує достатньо чіткі логічні моделі та методи, що можуть бути використані для моделювання реальних процесів, або розв’язання цілого комплексу задач, що виникають у процесі діяльності підприємств. Інструментарій та задачі комбінаторної оптимізації можна використовувати для отримання певних економічних ефектів на підприємствах, а саме - найпопулярнішу та найважливішу задачу комівояжера та методи її вирішення можна використовувати задля оптимізації транспортних витрат – складати оптимальні шляхи перевезення та знаходити оптимальні маршрути і, таким чином, скорочувати витрати на транспортування вантажів, скорочувати час на виконання замовлень, підвищувати робочу продуктивність та заощаджувати матеріальні ресурси підприємства. Тому доцільно використовувати комбінаторну оптимізацію для дослідження та оптимізації різноманітних сучасних процесів на підприємствах.

Задача комівояжера вважається однією з найважливіших задач теорії графів, класичним завданням дискретної оптимізації та дає можливість подання різних виробничих процесів на мові теорії графів і вміння вирішити сформульовану математичну задачу для знаходження оптимальної стратегії ведення господарства, заощадити ресурси, виконати поставлене завдання в

більш короткі терміни. Значний обсяг робіт з цієї проблематики свідчать про необхідність проведення подальших досліджень питань оптимізації транспортних витрат, зокрема із використанням методів комбінаторної оптимізації.

Таким чином у першому розділі кваліфікаційної роботи були розглянуті комбінаторна оптимізація та її методи для скорочення транспортних витрат, сформульоване та розглянуте завдання комівояжера та класифікація його різновидів.

РОЗДІЛ 2. АНАЛІЗ МОДЕЛЕЙ ТА МЕТОДІВ ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ ОПЕРАЦІЙ

2.1 Аналіз методів вирішення задачі комівояжера

Методи розв'язання задачі комівояжера досить різноманітні і розрізняються застосовуваним інструментарієм, точністю знаходять рішення і складністю необхідних обчислень.

Відомі методи розв'язання поділяють на групи, які можна комбінувати. Точні методи при наявності достатньої кількості часу знаходять гарантовано оптимальний шлях. Евристічні методи знаходять рішення, що можуть бути значно гіршими ніж оптимальні але за менший час.

До точних методів належать:

1) Повний перебір

2) Метод гілок і меж

Усі ефективні, а саме ті що скорочують повний перебір методи розв'язання задачі комівояжера - методи евристичні. У більшості евристичних методів знаходиться не найефективніший маршрут, а наближене рішення.

До евристичних методів належать:

1) Жадібний алгоритм та метод найближчого сусіда.

2) Мурашиний алгоритм

Також можна виділити імовірнісні методи. Ці методи здійснюють випадкові зміни шляху в очікуванні отримання найбільш короткого шляху. З цього класу методів відзначимо:

1) Алгоритм симуляції відпалу

2) Генетичний алгоритм

3) Метод мінімального остовного дерева

Розглянемо наведені методи більш детально.

Метод грубої сили

Повний перебір або метод грубої сили відноситься до класу методів пошуку рішення вичерпуванням варіантів і полягає в послідовному розгляді всіх можливих маршрутів і виборі з них оптимального. Будь-яке завдання з класу NP можна вирішити повним перебором. Хоча метод грубої сили простий і точний, але неефективний при великій кількості точок.

Ідея методу доволі проста: перебираються всі рішення і з них вибирається рішення або безліч рішень, що відповідає умові завдання. У завданні комівояжера, відповідно, потрібно зі всіляких варіантів об'їзду пунктів вибрати маршрут, що займає найкоротший час або мінімальний за вартістю маршрут.

Величезною перевагою методу повного перебору перед іншими методами вирішення Завдання комівояжера є гарантованість знаходження найкращого маршруту. Інші методи радять лише «гарний» маршрут, який зовсім не обов'язково є кращим. Крім того, до переваг методу відноситься простота його програмної реалізації.

Для реалізації методу повного перебору досить виробти генерацію всіх перестановок заданого числа елементів. Зробити це можна декількома способами. Наприклад лексикографічною перестановкою.

Нехай є деякий алфавіт і набори символів цього алфавіту – слова. Букви в алфавіті впорядковані. Наприклад, в українському алфавіті порядок букв наступний: а>б>я. Якщо впорядковані букви алфавіту, то можна впорядкувати і слова. Наприклад дано слово $w = (w_1, w_2,...,w_m)$, що складається з букв $l_1,l_2,...,l_m$, і слово $x = (x_1,x_2,...,x_b)$. Тоді якщо $w_1>x_1$, то і $w > x$. Якщо ж $w_1 = x_1$, то порівнюють другі літери і. т. д.. Такий порядок слів і називається лексикографічним.

Алгоритм лексикографічної перестановки [2] виглядає таким чином:

Нехай задана деяка перестановка 5-7-9-8-6. Потрібно рухатися по перестановці справа наліво, поки вперше не буде виявлено число, менше, ніж попереднє. Припустимо, що знайдене число розташовується на позиції P_{i-1}.

Міняємо знайдене число місцями з найменшим з великих чисел, які розташовані правіше позиції P_{i-1}. Після чого, числа правіше позиції P_{i-1} необхідно впорядкувати за зростанням. В результаті виходить наступна перестановка: в прикладі це 5-8-6-7-9, за нею слід 5-8-6-9-7 і. т. д.

Після генерації перестановки обчислюють суму проїзду між пунктами. Після того як згенеровано усі перестановки і пораховані суми проїзду для них, вибирається маршрут, відповідний мінімального часу об'їзду пунктів.

Істотний недолік цього методу залежить від кількості всіх можливих рішень задачі. Якщо простір рішень дуже великий, то повний перебір може не дати результатів протягом великої кількості часу, цей процес може займати роки. Симетрична задача комівояжера з *n* відвідуваних пунктів вимагає при повному переборі розгляду *(n-1)!* тур. Таким чином його застосування стає скрутним через значні витрати часу і ресурсів на перебір величезної кількості варіантів вирішення завдання.

Жадібні алгоритми і метод найближчого сусіда.

Жадібні алгоритми засновані на знаходженні локально оптимальних рішень на кожному етапі обчислень і допущенні, що знайдене таким чином підсумкове рішення буде глобально оптимальним. Тобто на кожній ітерації вибирається краща ділянка шляху, яка включається в підсумковий маршрут.

Метод простий, але його великий недолік в тому, що може виникнути ситуація, коли виявиться, що початкова і кінцева точки маршруту рознесені далеко один від одного і їх доведеться з'єднувати довгим відрізком шляху, що значно знизить ефективність рішення. До жадібним алгоритмів відносяться: метод найближчого сусіда, модифікований метод найближчого сусіда, метод найдешевшого включення і т. д.

Ідея алгоритму найближчого сусіда заснована на простому евристичному правилі: якщо відвідувати найближчий пункт на кожному кроці, то маршрут вийде досить хорошим в цілому. Перед комівояжером ставиться

завдання відвідувати найближчий з ще не відвіданих пунктів. В даному алгоритмі існують два важливих обмеження:

1. Недопущення повторного заїзду в пункт пов'язане з необхідністю знаходження гамільтонова циклу. Циклу, в якому всі пункти відвідуються один раз.

2. Недопущення передчасного повернення у вихідний пункт. Цей пункт необхідний для запобігання зациклення алгоритму і його коректної роботи.

Блок-схема роботи алгоритму зображена на рисунку 2.1

Рис. 2.1 Блок-схема алгоритму методу найжближчого сусіда

Метод мінімального остовного дерева

В основі алгоритму лежить твердження: «якщо справедлива нерівність трикутника, то для кожного ланцюга вірно, що відстань від початку до кінця ланцюга менше або дорівнює сумарної довжини всіх ребер ланцюга».

Це узагальнення переконання, що пряма коротше кривої. Дерев'яний алгоритм для вирішення. Спочатку будується на вхідній мережі завдання комівояжера найкоротше остовне дерево і подвоюються всі його ребра. В результаті отримують граф, зв'язний з вершинами, що мають тільки парні ступені. Потім будується Ейлерів цикл, починаючи з першої вершини, цикл задається переліком вершин. Проглядається перелік вершин, починаючи з першої, і закреслюється кожна вершина, яка повторює вже зустрінуту в послідовності. Залишиться тур, який і є результатом алгоритму.

Доведено, що цей алгоритм помиляється менш ніж в два рази, тому такі алгоритми називають приблизними, а не просто евристичними.

Для знаходження мінімального остовного дерева існують алгоритми Прима, Крускала і Борувки.

Алгоритм Прима [94] працює наступним чином: на вхід алгоритму подається зв'язний неорієнтований граф і для кожного його ребра задається вартість. У першу чергу потрібно взяти довільну вершину і знайти ребро, яке володіє найменшою вартістю і є інцидентним даній вершині. Знайдене ребро і з'єднувані їм дві вершини утворюють дерево. Далі розглядають ребра графа, один кінець яких є вершиною, що належить дереву, а інший - ні. З цих ребер обирають найменше за вартістю. Обиране на кожному кроці ребро приєднують до дерева. Зростання дерева відбувається до тих пір, поки не будуть використані всі вершини вихідного графа. Результатом роботи алгоритму є остовне дерево мінімальної вартості.

Алгоритм Крускала [96] працює наступним чином: спочатку поточну множину ребер встановлюють порожньою. Далі проводиться наступна операція: з усіх ребер, додавання яких до вже наявної множини не викличе появу в ній циклу, обирають ребро мінімальної ваги і додають до вже наявної

множини. Ця операцію проводять до тих пір, поки це можливо. Алгоритм завершується, коли таких ребер більше немає. Підграф даного графа, що містить всі його вершини і знайдену множину ребер є його остовним деревом мінімальної ваги.

Алгоритм Борувки полягає в декількох ітераціях, кожна з яких полягає в послідовному додаванні ребер до остовному лісі графа, до тих пір, поки ліс не перетвориться в дерево.

Алгоритм можна описати так [4]:

Спочатку, нехай *T* є порожньою множиною ребер, яка являє собою остовний ліс, в який кожна вершина входить в якості окремого дерева.

Поки *T* не є деревом, поки число ребер в *T* менше, ніж *V-1*,

де *V*-число вершин в графі: для кожного дерева в остовному лісі в підграфі з ребрами *T* знайдемо найдешевше ребро, що зв'язує це дерево з деяким іншим. Передбачається, що ваги ребер різні або додатково впорядковані так, щоб завжди можна було знайти єдине ребро з мінімальною вагою. Всі знайдені ребра додають в безліч *T*.

Отримана множина ребер T буде мінімальним остовним деревом для вхідного графа.

Метод імітації відпалу

Назва даного алгоритму пов'язана з методами імітаційного моделювання, заснованими на техніці Монте-Карло. Дослідження кристалічної решітки і поведінки атомів при повільному охолодженні тіла призвело до появи на світ імовірнісних алгоритмів, які виявилися надзвичайно ефективними в комбінаторній оптимізації. Алгоритм імітації відпалу грунтується на імітації фізичного процесу, який відбувається при кристалізації речовини з рідкого стану в тверде, в тому числі при відпалі металів. Сьогодні цей алгоритм є популярним як серед практиків завдяки своїй простоті, гнучкості та ефективності, так і серед теоретиків, оскільки для даного

алгоритму вдається аналітично досліджувати його властивості і довести асимптотичну збіжність.

Алгоритм роботи методу імітації відпалу [97]:

1) Вибір початкового рішення та початкової температури

2) Оцінка початкового рішення

3) Основний крок алгоритму:

- випадкова зміна поточного рішення;

- оцінка зміненого рішення;

- критерій допуску.

4) Зменшення температури.

Якщо температура більше деякого порога, то перехід до кроку 3.

На початку алгоритму слід вибрати початкову і мінімальну температуру. Початкова температура t_{max} - це початок відпалу, а мінімальна температура t_{min} – це кінець відпалу

Задається довільний початковий стан системи S_0, тим самим визначається функція для нового стану. Далі слід порівняти два значення цільової функції - поточне та після прийняття нового значення – ES_0 и ES_i відповідно. Якщо обчислене значення цільової функції ES_i виявилося менше, ніж поточне, тоді зберігаємо поточне в якості цільової функції

$$ES_i < E\ S_0 \rightarrow S_0 = S_i. \quad (2.1)$$

Дане правило застосовується для завдання на пошук глобального мінімуму. Якщо завдання на пошук максимуму, то формула буде виглядати так:

$$ES_i > ES_0 \rightarrow S_0 = S_i. \quad (2.2)$$

Ймовірність прийняття поточного значення обчислюється за формулою:

$$P = e^{\frac{-\delta E}{Ti}} \quad (2.3)$$

Де P – ймовірність прийняття значення;

$\delta E = ES_0 - ES_i$,

S_0 – поточний стан,

S_i – новий прийнятий стан;

T_i – поточна температура.

Якщо обчислена ймовірність більше певного значення, тоді зберігають новий стан системи $S_0 = S_i$. Після обчислюється нове значення температури T_i, вона може знижуватися в залежності від закономірності даної задачі, наприклад лінійної. Лінійну закономірність можна представити у вигляді формули:

$$T_i = T_{i-1} * \frac{r}{j}, \quad (2.4)$$

Де *j – крок ітерації;*

r – коефіцієнт, який налаштовується та зазвичай приймає значення від 0,8 до 0,99

Якщо виконується умова $T_i < T_{min}$, тоді алгоритм завершується, інакше переходимо до вибору нового стану. Блок-схема алгоритму відпалу зображена на рисунку 2.2

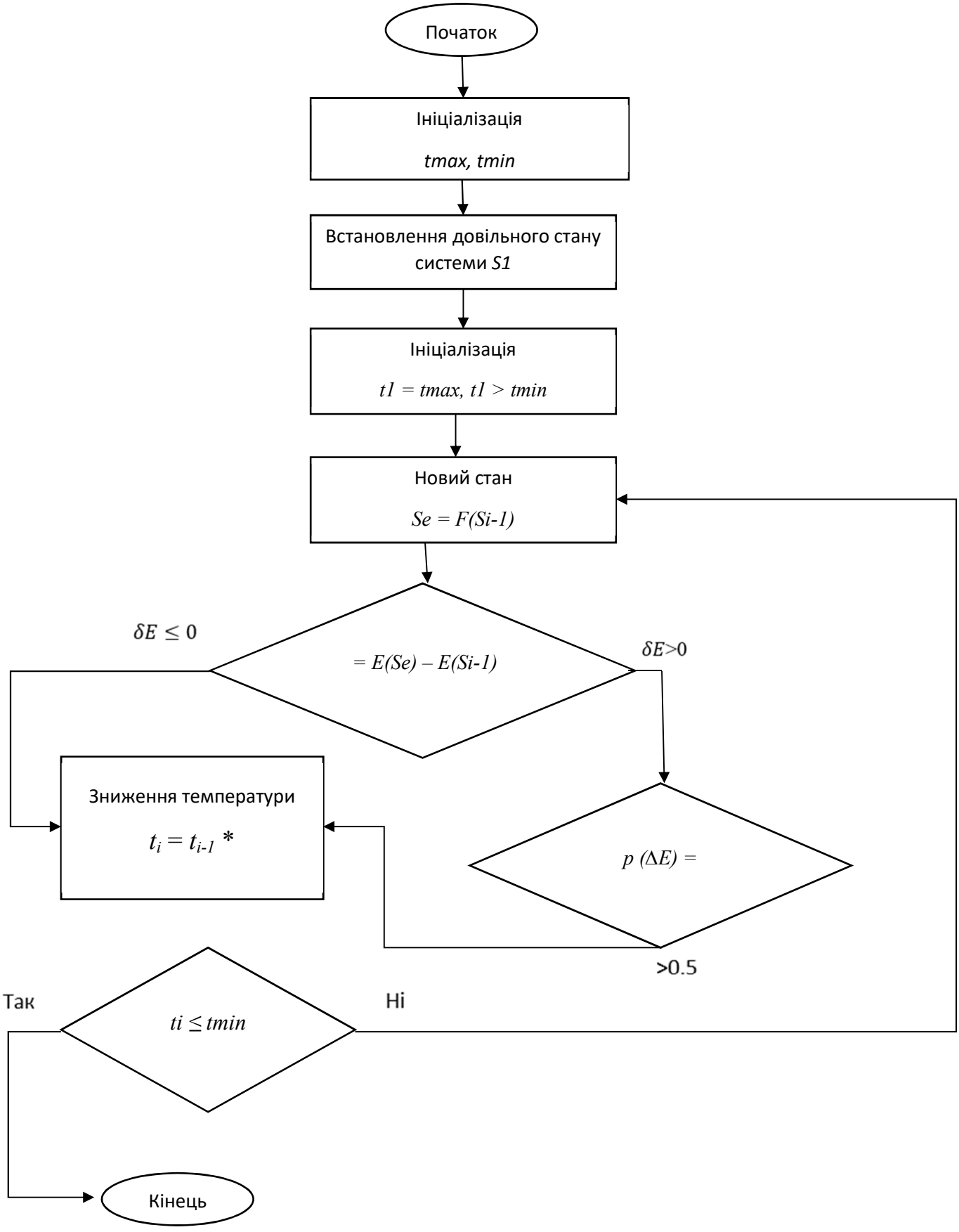


Рисунок 2.2 Блок схема алгоритму імітації відпалу

Мурашиний алгоритм

Мурашиний алгоритм - евристичний метод, заснований на моделюванні поведінки мурах, які шукають шляхи від своєї колонії до джерел їжі. Першу версію такого алгоритму запропонував доктор наук Марко Доріго в 1992 році. Цей метод дозволяє знайти хороше, але не обов'язково оптимальне рішення за поліноміальний час.

В основі даного методу лежить поведінка мурах у реальному світі. У пошуках їжі мураха проходить випадковий шлях від гнізда до місця розташування їжі, а після, повертаючись, залишає за собою слід з феромона. Цей феромон приваблює інших мурах, що знаходяться поблизу до сліду з феромона. Таким чином вони пройдуть по даному сліду, тим самим повертаючись в гніздо і зміцнюючи феромонну стежку. Якщо ж існує два і більше маршруту від їжі до гнізда, то по більш короткому встигнуть пройти більше мурах, ніж по довгому. І тоді короткий шлях стає привабливим для мурах через велику кількість феромона. Довгі ж шляхи в результаті зникнуть через випаровування.

Блок-схему мурашиного алгоритму зображено на рисунку 2.3

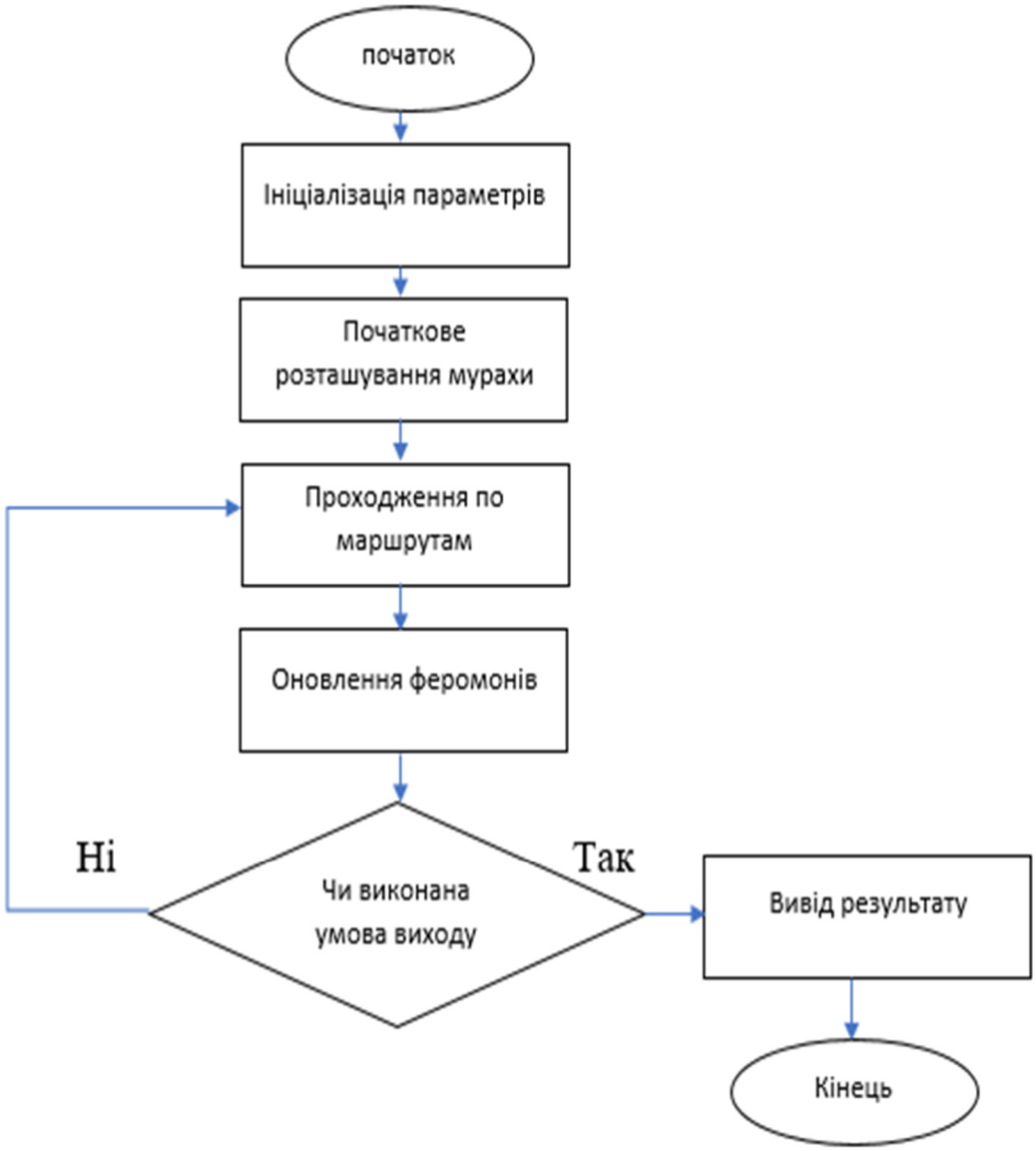


Рисунок 2.3 Блок-схема мурашиного алгоритму.

Генетичний алгоритм

Генетичний алгоритм - ще один евристичний метод, що полягає у випадковому підборі і комбінуванні вихідних параметрів з використанням механізмів імітують природний відбір в процесі еволюції - спадкування, мутації, кросинговер.

Засновником генетичних алгоритмів вважається Джон Холланд і його праця «Адаптація в природних і штучних системах».

Генетичний алгоритм належить до евристичних алгоритмів пошуку і використовується для вирішення завдань оптимізації та моделювання шляхом випадкового підбору, комбінування і варіації параметрів. Особливістю

алгоритму є використання механізму, що нагадує процеси еволюції – використання спеціального оператора схрещування, який виробляє операцію рекомбінації рішень-кандидатів, роль якої аналогічна ролі схрещування в живій природі.

На початку роботи алгоритму необхідно задати цільову функцію, функцію пристосованості для популяції, і початкову популяцію, яка створюється випадковим чином. Потім визначається схрещування - береться два гени різних особин, згодом вони називаються батьками, схрещуваються таким чином, щоб нащадки від даних батьків успадкували риси кожного з батьків.

Наступним кроком відбувається мутація отриманих нащадків і етап відбору. Відбір відбувається залежно від функції приналежності з особин, отриманих на попередніх етапах, виживає тільки частина.

При застосуванні генетичного алгоритму для вирішення завдання комівояжера, замість функції пристосованості, для кожної особини використовується довжина шляху об'їзду міст.

Блок-схема генетичного алгоритма зображено на рисунку 2.4

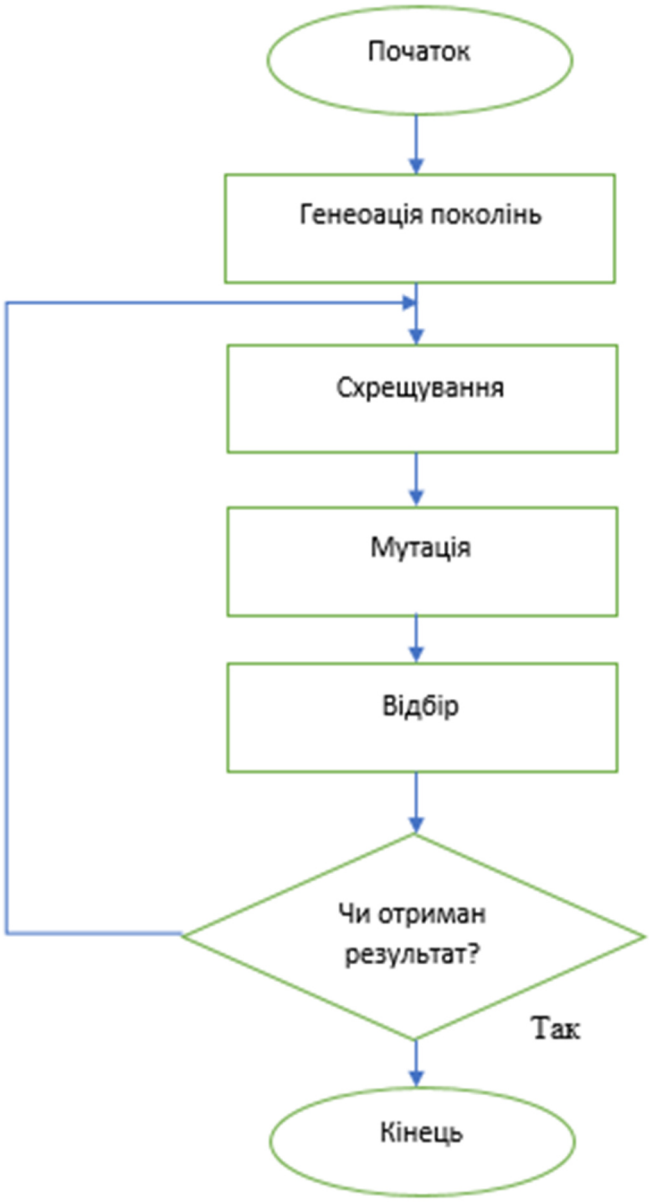


Рисунок 2.4 – Блок-схема генетичного алгоритму

Метод гілок і меж

Метод гілок і меж - один з методів дискретної оптимізації, що є розвитком методу повного перебору, але відрізняється від нього відсівом в процесі обчислення підмножин неефективних рішень. Для прискорення перебора методом гілок та меж використовують відсів підмножин допустимих рішень, що не містять оптимальних рішень [3].

Широко використовуваний варіант пошуку з поверненням, фактично є лише спеціальним окремим випадком методу пошуку з обмеженнями. Обмеження в даному випадку ґрунтуються на припущенні, що на безлічі

можливих і часткових рішень задана деяка функція ціни і що потрібно знайти оптимальне рішення, тобто рішення з найменшою ціною. Для застосування методу гілок і меж функція ціни повинна володіти тією властивістю, що ціна будь-якого часткового рішення не перевищує ціни будь-якого розширення цього часткового рішення.

В основі методу гілок і меж лежить ідея послідовного розбиття безлічі допустимих рішень на підмножини.

На кожному кроці методу елементи розбиття піддаються аналізу – чи містить дана підмножина оптимальне рішення чи ні. Якщо розглядається задача на мінімум, то перевірка здійснюється шляхом порівняння нижньої оцінки значення цільової функції на даній підмножині з верхньою оцінкою функціоналу. В якості оцінки зверху використовується значення цільової функції на деякому допустимому рішенні. Допустиме рішення, що дає найменшу верхню оцінку, називають рекордом. Якщо оцінка знизу цільової функції на даному підмножині не менше оцінки зверху, то підмножина, що розглядається не містить рішення краще рекорду і може бути відкинута. Якщо значення цільової функції на черговому рішенні менше рекордного, то відбувається зміна рекорду. Кажуть, що підмножина рішень переглянута, якщо встановлено, що вона не містить рішення краще рекорду. Якщо переглянуті всі елементи розбиття, алгоритм завершує роботу, а поточний рекорд є оптимальним рішенням. В іншому випадку серед непроглядених елементів розбиття вибирається множина, що є в певному сенсі перспективною. Вона піддається розбиттю (розгалуженню). Нові підмножини аналізуються за описаною вище схемою. Процес триває до тих пір, поки не будуть переглянуті всі елементи розбиття. Блок-схему методу гілок та меж зображено на рисунку 2.5

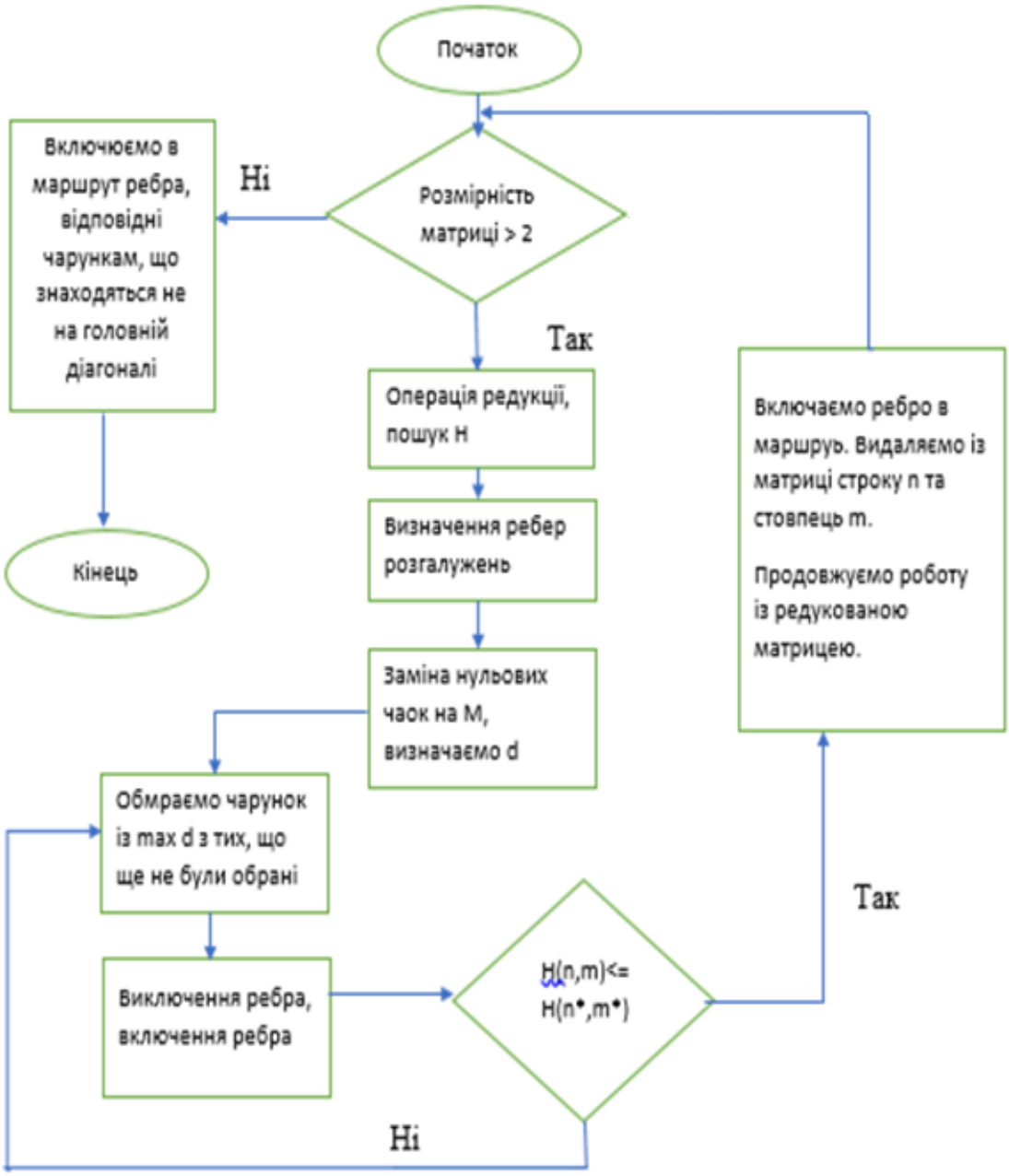


Рис. 2.5 – Блок-схема алгоритму гілок та меж

2.2 Вибір методу оптимізації транспортних витрат

Слід зазначити, що метод гілок і меж набув найбільшого поширення серед схем неявного перебору в основі якого лежить ідея послідовного розбиття безлічі допустимих рішень.

В якості обґрунтування вибору методу для реалізації варто привести його великий успіх і популярність застосування. Це пояснюється тим, що автори методу першими звернули увагу на широту його можливостей і відзначили

важливість використання специфіки завдання. Так само підставою є дані отримані з таблиць порівняння методів для прийняття рішення запропоновані вище: метод є точним, метод є простим для розуміння та досить простий в програмній реалізації. У порівнянні з точним методом грубої сили, метод гілок і меж можна використовувати при невеликій, але значно більшій кількості точок та для асиметричних завдань. Порівняння деяких методів вирішення задачі комівояжера представлене у таблиці 2.1

Таблиця 2.1

Порівняння деяких методів вирішення задачі комівояжера

	Критерій трудомісткості	Критерій використання пам'яті	Якість рішення
Метод гілок та меж	O(n lg2n)	~(2n3)	точний
Меторд грубої сили	O(n!)	n2	точний
Генетичний алгоритм	O(t×n× $m2$)	~n2	наближений
Метод найближчого сусіда включення	O(n)	~n2	наближений

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі кваліфікаційної роботи були проаналізовані моделі та методи вирішення завдання комівояжера та обрано метод для подальшої реалізації.

Методи вирішення Завдання комівояжера досить різноманітні і розрізняються застосовуваним інструментарієм, точністю рішення, що знаходиться і складністю необхідних обчислень.

Різні ситуації і проблеми вимагають застосування спеціальних методів, як окремих, так і комбінованих для досягнення необхідного оптимального результату. Однак саме метод гілок і меж для оптимізації транспортних витрат, а саме для складання оптимального маршруту розвезення вантажів дає в результаті рішення близьке до оптимального за малий, в порівнянні з іншими методами, час. Метод гілок і меж вважається універсальним методом, так як він добре підходить для вирішення асиметричного завдання комівояжера. У той час як інші методи пристосовані в основному для вирішення симетричних завдань. Незважаючи на недолік методу гілок і меж, збільшення часу обчислень при збільшенні розмірності задачі, можна вважати, що алгоритми методу є надійним засобом вирішення мінімаксних задач про шляхи або призначення, що зустрічаються в практичних дослідженнях.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ МЕТОДІВ ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ ОПЕРАЦІЙ

3.1 Характеристика підприємства «Інтегровані системи безпеки»

ТОВ «Інтегровані системи безпеки» - це підприємство, що діє на основі статуту та надає широкий спектр послуг з IT-сфери, серед них:

1) Продаж комп’ютерної, аудіо, відео техніки, систем безпеки та відеонагляду.
2) Прокладка кабелів, роботи з монтажу пристроїв відео нагляду та безпеки.
3) Продаж комплектуючих частин для систем безпеки та відеонагляду.
4) Післяпродажне, сервісне та гарантійне обслуговування.
5) Ремонт комп’ютерної, периферійної, аудіо, відео техніки та пристроїв безпеки та відеонагляду.
6) Послуги виїзного IT-спеціаліста.
7) Продаж програмного забезпечення.
8) Розробка програмного забезпечення.
9) Монтаж та розгортання структурованих кабельних систем, налаштування мережевих пристроїв.

Клієнт – фізична або юридична особа, яка використовує продукцію або послуги підприємства.

Після продажне обслуговування– це комплекс організаційно-технічних заходів, спрямованих на забезпечення надійності та безпеки продукції, виконуваних протягом усього періоду експлуатації у вигляді сервісних послуг, що надаються клієнту.

Гарантійне обслуговування – виконання підприємством гарантійних зобов'язань заводу виробника за якістю і комплектності його продукції в гарантійний період експлуатації.

Післягарантійне обслуговування - виконання підприємством різних видів ремонту продукції після закінчення дії гарантійних зобов'язань.

Не гарантійне обслуговування – виконання підприємством обслуговування продукції в гарантійний період експлуатації, але з незалежних від заводу виробника не гарантійних випадків виходу продукції з ладу.

Гарантійний ремонт – комплекс робіт, пов'язаних з виконанням гарантійних зобов'язань, спрямованих на усунення несправностей і відмов в продукції для відновлення втраченої в період гарантійного періоду працездатності в межах експлуатаційних характеристик, встановлених у технічній документації.

Післягарантійний ремонт – комплекс робіт, що здійснюється за межами гарантійного терміну експлуатації, спрямованих на усунення відмов і відновлення працездатності продукції на комерційній основі.

Не гарантійний ремонт – комплекс робіт, здійснюваний в межах гарантійного терміну експлуатації по які гарантійних випадків виходу з ладу продукції на комерційній (платною) основі.

Послуги виїзного IT-спеціаліста - консультування с питань IT, послуги з налаштування, установки, розробки, підтримки та програмного супроводу із виїздом спеціалісту до місця надання послуг. IT- аутсорсинг.

Налаштування мережевих пристроїв - налаштування комутаторів, роутерів, мережевих принтерів, мережевих фабрик друку, мережевих сканерів, пристроїв, що мають підключення до ip-мереж, тощо.

Підприємство в своєму складі має такі відділи:

- відділ по роботі з клієнтами;
- технічний відділ;
- бухгалтерія;
- транспортно-логістичний відділ;
- адміністрація.

Відділ по роботі з клієнтами приймає заявки, займається пошуком та залучення нових клієнтів, опрацьовує закази клієнтів, встановлює завдання, формує рахунки для клієнтів, передає завдання до технічного відділу, а також регулює можливі конфліктні ситуації, що виникли при наданні послуг,

сервісному та гарантійному обслуговуванні комп’ютерної, аудіо, відео техніки та систем безпеки і відеонагляду.

До основних функцій відділу по роботі з клієнтами відносяться наступні:

- розгляд звернень клієнтів;
- оформлення відповідної документації, створення рахунків;
- надання інформаційних послуг;
- розробка та контроль виконання додаткових заходів щодо конфліктних ситуацій між клієнтом і підприємством;
- пошук та залучення нових клієнтів;

Технічний відділ відповідає за монтажні роботи та надання сервісу, ремонти.

Даний відділ представлений інженерами, що забезпечують різні види монтажних та ремонтних робіт комп'ютерної, аудіо, відео техніки, систем безпеки та відеонагляду.

Бухгалтерія. Даний відділ займається різними питаннями, що стосуються фінансових оборотів на підприємстві, а саме:

- обробка первинної документації;
- аналіз інформації, що надходить від інших відділів;
- ведення взаєморозрахунків з постачальниками і клієнтами;
- ведення податкового обліку.

Транспортно-логістичний відділ виконує транспортування вантажів та продукції, перевезення спеціалістів до місць виконання робіт, розподіл та зберігання продукції та товарів на складах. До транспортно-логістичний відділу належать транспортні засоби та складські приміщення. Робота транспортно-логістичного відділу здійснюється за допомогою водіїв та комірників.

Адміністрація підприємства забезпечує контроль роботи всіх відділів підприємства, встановлює плани роботи, проводить управління всіма процесами. Структуру підприємства зображено на рисунку рисунку 3.1.

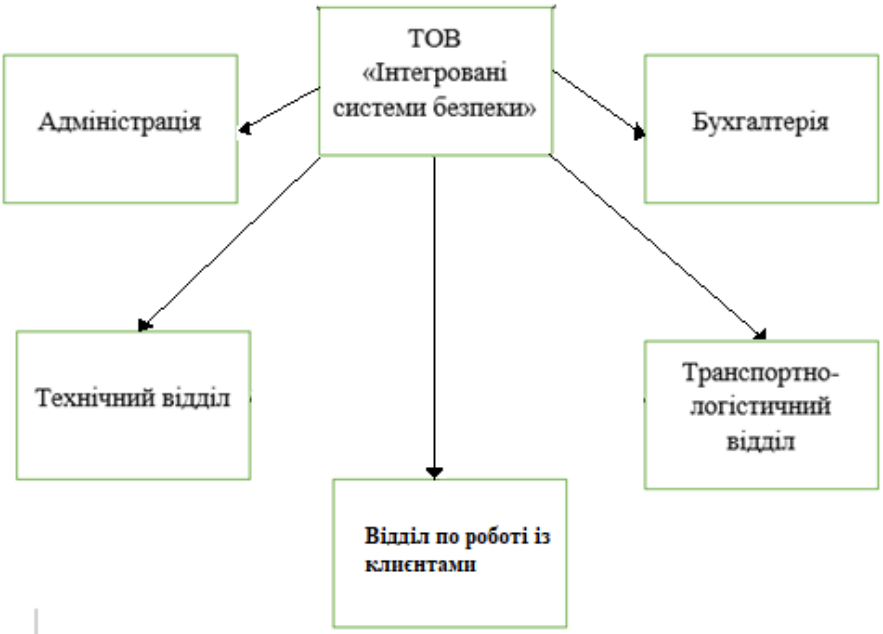


Рис. 3.1 Структура підприємства

У кожному з відділів числиться певний набір співробітників. Ставлення співробітників до відділів зображено на рисунках 3.2 – 3.7.

Рис. 3.2 Ієрархія співробітників відділу адміністрації

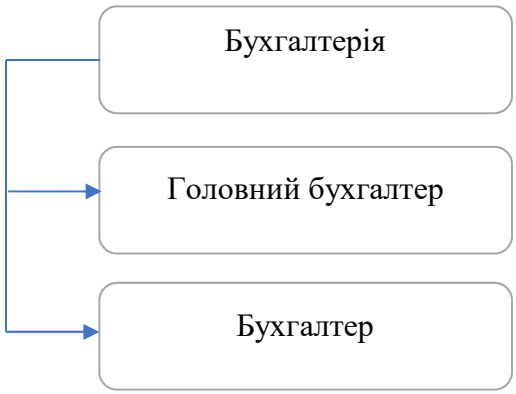


Рис. 3.3 Ієрархія співробітників бухгалтерії

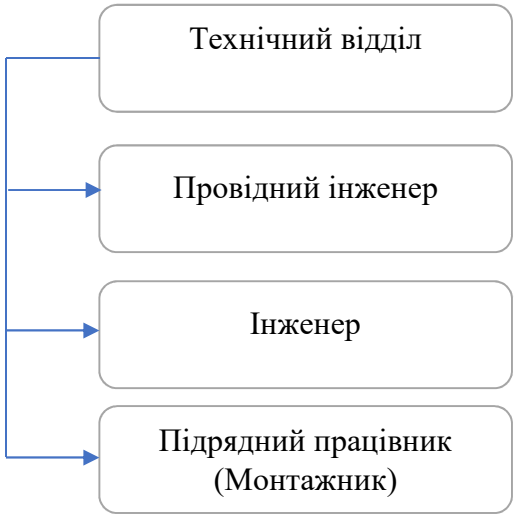


Рис. 3.4 Ієрархія співробітників технічного відділу

Рис. 3.5 Ієрархія співробітників відділу по роботі з клієнтами

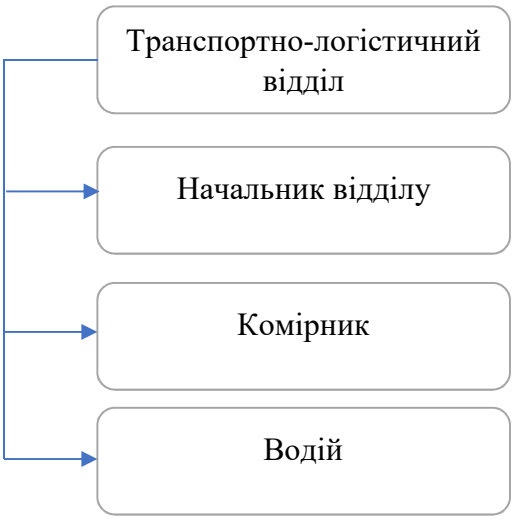


Рис. 3.6 Ієрархія співробітників транспортно-логістичного відділу

Рисунки ілюструють відділи підприємства і основні посади співробітників.

Перелік регламентуючих документів, за допомогою яких здійснюється діяльність підприємства.

- договори (субпідряди, договори з перевізниками, постачальниками та ін.);
- внутрішні і зовнішні накази, постанови;
- інструкції з експлуатації, технічного обслуговування та ремонту;
- інструкції для сервісних організацій;
- сертифікати, що підтверджують право на обслуговування обладнання.

Всі наведені вище регламентуючі документи надають можливість підприємству, і, зокрема, цільовим відділам, здійснювати свої функції.

Договори (субпідряди, договори з перевізниками, постачальниками та ін.) Регламентують значну частину діяльності підприємства. Основний напрямок – регулювати зовнішню політику, в тому числі взаємини з іншими підприємствами: з фірмами-виробниками, клієнтами та ін. Другою же основною функцією є декларування повного або часткового виконання будь-яких робіт.

Внутрішні і зовнішні накази, постанови регулюють внутрішню політику підприємства, а також деякі аспекти зовнішньої.

Інструкції по експлуатації, технічного обслуговування та ремонту обладнання необхідні для виконання функції першорядної важливості для підприємства, а саме ремонту комп'ютерного, аудіо, відео обладнання систем безпеки та відеонагляду. У них детально описана кожна модель обладнання будь-якого виду, наведено перелік стандартних проблем, що виникають з моделлю, а так само способи вирішення проблем.

Сертифікати, які підтверджують право на обслуговування обладнання, підтверджують право спеціалістів займатися різними роботами з тими чи іншими видами обладнання.

Також розглядаючи підприємство потрібно розглянути процеси від прийняття замовлення до виконання замовлення чи надання послуги. Відбувається це таким чином – перш за все клієнт залишає замовлення у відділі по роботі з клієнтами, працівники відділу допомагають сформулювати технічне завдання, формують договори та рахунки. Після затвердження рахунки відправляються клієнтам, а технічне завдання та договір до технічного та транспортно-логістичного відділу. Якщо не передбачена передоплата, то відділи виконують свою частину роботи. Технічний відділ надає послуги, або передає продукцію далі у транспортно-логістичний відділ. Після транспортно-складських робіт відділ здає звіт та акти про виконані роботи. Якщо передбачена передоплата, то відділи виконують свою частину роботи після підтвердження бухгалтерського відділу про оплату.

За допомогою технології DFD (методологія діаграм потоків даних) створимо контекстну діаграму функціональної моделі процесу обробки замовленя (рис. 3.7).

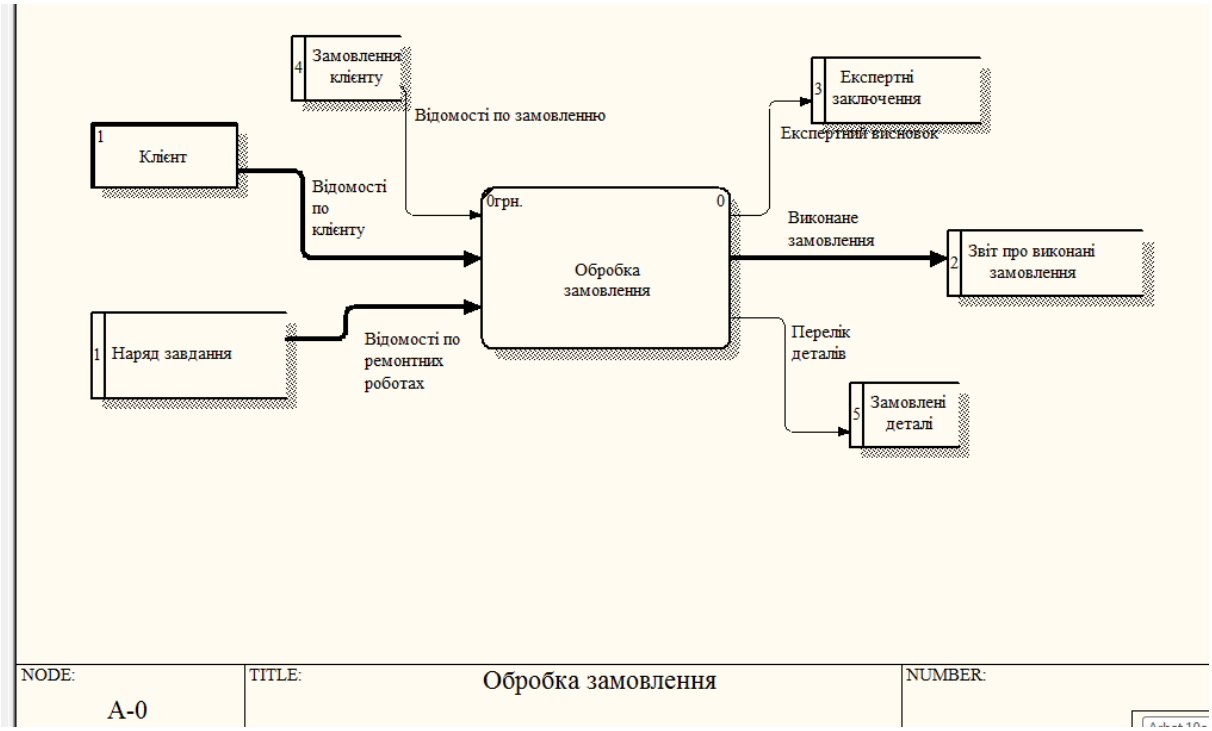


Рис. 3.7 Контекстна діаграма в методології DFD

В контекстній DFD діаграмі (рис.3.7) «Клієнт» представляє зовнішню сутність, а «Звіт про виконані замовлення», «Замовлені деталі», «Замовлення клієнту», «Експертні висновки» – сховища даних (журнали, відомості, картки, папки з документами).

Наступним кроком слід здійснити декомпозицію діаграми першого рівня (рисунок 3.8).

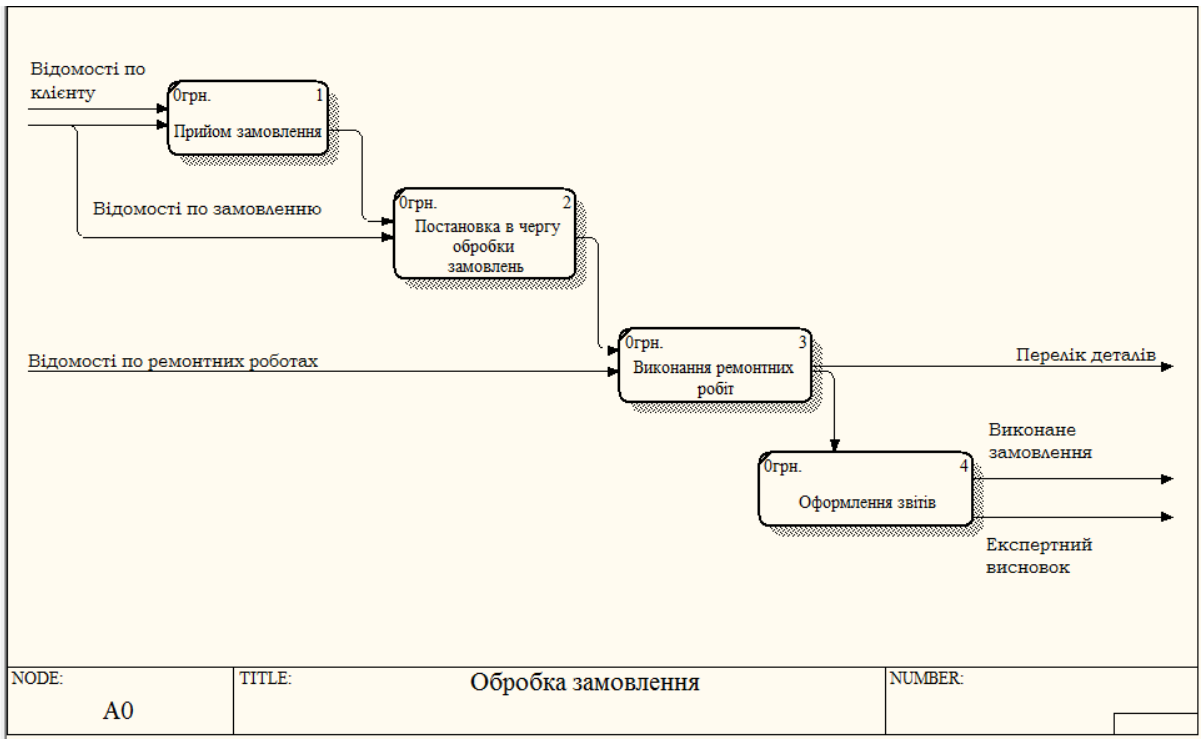


Рис 3.8 DFD- діаграма декомпозиції першого рівня

На діаграмі з'явилися наступні процеси: «Прийом замовлення», «Постановка в чергу обробки замовлень», «Виконання ремонтних робіт», «Оформлення звітів», які ілюструють життєвий цикл виконання замовлення. Також на діаграмі зображені сховища даних з попередньої контекстної діаграми. Суттєвим є те, що існує черга обробки замовлень, але вплив на неї можуть оказати тільки начальник відділу по роботі з клієнтами.

3.2 Постановка задачі оптимізації транспортних витрат підприємства «Інтегровані системи безпеки»

Постановка задачі оптимізації транспортних витрат для ТОВ «Інтегровані системи безпеки» наступна:

Згідно із проектом підприємство має відвідати визначені точки, що зазначені у договорі та виконати встановлений план робіт. Всього існує 60 точок, які обслуговуються бригадами підприємства, склад кожної бригади - 1 водій автотранспортного засоба та 4 монтажника.

Потрібно сформувати таблиці адрес за районами, що будуть розподілені між бригадами, сформувати матриці відстаней та вирішити задачу

комівояжера обраним методом оптимізації транспортних витрат. Отримати оптимальний план маршрутів бригад таким чином, щоб шлях, який проїде кожна бригада був мінімальний. Точки адрес для виконання проекту наведені у таблиці 3.1

Таблиця 3.1

Місця встановлення блоків сповіщення.

№	Заклад	Населений пункт	Вулиця/провулок	будинок	місце встановлення	Кіль-кість
1	Спеціалізована школа №5	Маріуполь	вул. Київська	72	Вестибюль (1 поверх)	1
2	ЗОШ №15	Маріуполь	Новоросійська буд.	12	І поверх (хол)	1
3	ЗОШ №20	Маріуполь	вул. Ливарна	4	1 поверх (коридор)	1
4	ЗОШ №21	Маріуполь	вул. Озеро Хасан	65	корпус №1 (коридор) ; корпус №2 (коридор)	1
5	ЗОШ №24	Маріуполь	проїзд Шопена	4	І поверх (хол)	1
…	…	…	…	…	…	…

3.3 Математична та алгоритмічні моделі методу гілок та меж оптимізації транспортних операцій

Як відомо, завдання комівояжера є NP-важким завданням, тому точні алгоритми його вирішення не можуть мати тимчасову складність менше експоненціальної. Рішенням є один з $(n-1)!$ циклів обходу по всіх n вершинах повного зваженого графа, сума довжин ребер в якому мінімальна[12]. Такий граф зазвичай задається матрицею невід'ємних відстаней (довжин ребер графа для кожної пари вершин) розміром $n \times n$, в якій діагональні елементи покладаються рівними дуже великій величині, теоретично нескінченності[12]. У найзагальнішому випадку така матриця асиметрична, а для її елементів не виконується нерівність трикутника. Одним з відомих ранніх алгоритмів точного рішення задачі комівояжера для загального випадку є алгоритм

Літтла, заснований на методі гілок і кордонів, який будує дерево рішень для перебору варіантів маршруту з відсіканням.

Алгоритм Літтла для вирішення Завдання комівояжера формулюється у наступному вигляді[25]:

1. У кожному рядку матриці $C = |cij|$ потрібно знайти мінімальний елемент di та записати його у окремий стовпець:

$$di = c_{ij,} \text{ (3.1)}$$

знайдені значення di називають константами приведення. Константи записують у окремий стовпець.

Далі відбувається процес редукції строк. Для цього віднімаємо мінімальний елемент з усіх елементів відповідного рядка. Отримана, наведена за рядками, матриця з елементами:

$$c'_{ij} = c_{ij} - d_{ij} \text{ (3.2)}$$

2. Таким самим чином якщо потрібно проводять процес редукції для кожного стовпця.

У кожному стовпці матриці C' обираємо мінімальний елемент d_j ,

$j = 1, \dots, n$ і віднімемо його з усіх елементів відповідного стовпця.

У результаті редукції за рядками та стовпцями отримаємо наведену по рядах і стовпцях матрицю, кожен рядок і стовпець якої містить хоча б один нуль:

$$C' = ||c_{ij} - c_{ij} - c_{ij}'||, \text{ (3.3)}$$

3. На наступному кроці треба підсумувати елементиd_i и d_j та знайти локальну нижню межу початкової точки:

$$\gamma = \sum_{i=1}^{n} d_i + \sum_{j=1}^{n} d_j, \text{ (3.4)}$$

Отримаємо константу приведення, яка буде локальною нижньою межею безлічі всіх допустимих гамільтонових контурів[26] (поточне значення є мінімальною довжиною маршрута на даному етапі), тобто:

$$\gamma = \varphi(\Omega^0) \leq z(X). \text{ (3.5)}$$

4. На цьому кроці потрібно знайти оцінки нульових чарунків для наведеної по рядках і стовпцях матриці. У подумках змінюємо нулі матриці знаком нескінченності та знаходимо суму мінімальних елементів рядка і стовпця, відповідних цьому нулю.

Для кожної нульової чарунки знайдемо її оцінку P_{ij} – ця оцінка є сумою мінімуму за рядком і стовпцем на перехресті яких знаходиться чарунок із нулем. За аналогією потрібно знайти оцінки для усіх інших нульових чарунків.

Записуємо оцінку в правому верхньому кутку клітини:

$$P_{ij} = \min_{j' \neq j} c_{ij'} + \min_{i' \neq i} c_{i'j}, \quad (3.6)$$

5. Обираємо дугу (i_0, j_0), для якої оцінка нульового елемента досягає максимального значення:

$$P_{i0j0} = \max_{c'ij=0} P_{ij}, \quad (3.7)$$

6. Розбиваємо множину всіх гамільтонових контурів Ω^0 на дві підмножини Ω^1_{i0j0} и $\Omega \frac{1}{i0j0}$.

Підмножина Ω^1_{i0j0} гамільтонових контурів містить дугу (i_0, j_0), а $\Omega \frac{1}{i0j0}$ - її не містить.

Для отримання матриці контурів Ω^1_{i0j0}, які включають в себе (i_0,j_0), викреслимо в матриці C'' строку i_o и стовпец j_o.

Щоб не утворився негамільтонов контур, замінимо симетричний елемент (i_0, j_0) знаком нескінченності.

7. Наводимо матрицю гамільтонових контурів Ω^1_{i0j0} .

Нехай h^1_{i0j0} - константа її приведення. Тоді нижню границю множини Ω^1_{i0j0} визначимо так:

$$\varphi\left(\Omega^1_{i0j0}\right) = \gamma + h^1_{i0j0}. \quad (3.8)$$

8. Знаходимо множину гамільтонових контурів $\Omega \frac{1}{i0j0}$, які не містять дугу (i_0,j_0).

Вилучення дуги (i_0, j_0) досягається заміною елемента C''_{i0j0} в матриці C'' на знак нескінченності.

9. Далі підлягає приведенню матриця гамільтонових контурів$\Omega\frac{1}{i0j0}$.

Нехай h^1_{i0j0} - константа її приведення. Нижня границя множини $\Omega\frac{1}{i0j0}$ визначиться так:

$$\varphi\left(\Omega\frac{1}{i0j0}\right) = \gamma + h^1_{i0j0}. \quad (3.9)$$

10. Нижні границі подмножини гамільтонових контурів Ω^1_{i0j0} и $\Omega\frac{1}{i0j0}$ порівнюють.

У випадку $\varphi\left(\Omega^1_{i0j0}\right) < \varphi(\Omega\frac{1}{i0j0})$ розгалуженню підлягає множина Ω^1_{i0j0}.

У випадку $\varphi\left(\Omega^1_{i0j0}\right) > \varphi(\Omega\frac{1}{i0j0})$ розбиттю підлягає множина $\Omega\frac{1}{i0j0}$.

Процес розбиття множин на підмножини супроводжується побудуванням дерева розгалужень.

11. Якщо після розгалужень отримано матрицю 2×2 - визначимо отриманий розгалуженням гамільтонів контур та його довжину.

12. Наступний етап зводиться до порівняння довжини гамильтонова контура із нижніми межами обірваних гілок. Коли довжина контуру не перевищує їх нижніх кордонів, то завдання вважають вирішеним. Інакше розвивають гілки підмножин з нижньою межею, меншою отриманого контуру, до тих пір, поки не отримають маршрут з меншою довжиною або не переконаються, що такого не існує.

3.4 Вибір IDE для розробки системи оптимізації транспортних витрат

Qt Creator-кроссплатформенне інтегроване середовище, призначене для розробки Qt-додатків на таких мовах програмування, як C, C++ і QML. У ньому включений зручний відладчик, а також різні візуальні засоби розробки інтерфейсу, що використовують QtWidgets і QML. Творці даного програмного

рішення ставили собі за мету створити засіб, значно спрощує розробку додатків за допомогою фреймворку Qt для різних платформ [100].

Qt Creator дозволяє виконувати вузькоспеціалізовані функції, наприклад, такі як налагодження додатків на QML і відображення даних в відладчику з контейнерів Qt [100]. 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 автодоповнення, підсвічування коду, а також можливість завдання стилів вирівнювання, постановки дужок і відступів [100].

Qt дозволяє запускати написане з його допомогою програмне забезпечення у більшості сучасних операційних системах 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.

Відмітна особливість - використання метаоб'єктного компілятора-попередньої системи обробки вихідного коду. Розширення можливостей забезпечується системою плагінів, які можливо розміщувати безпосередньо в панелі візуального редактора [98]. Також існує можливість розширення звичної функціональності віджетів, пов'язаної з розміщенням їх на екрані, відображенням, перемальовуванням при зміні розмірів вікна [98].

3.5 Інформаційна модель системи оптимізації транспортних витрат

Клас Algorithm

Властивості/поля:

char* name = "Algorithm"; // Назва алгоритма

std::vector<std::vector<int>> data; // Масив значень (матриця)

Методи:

Algorithm(); //Конструктор

Algorithm(std::vector<std::vector<int>>);//Конструктор з параметрами

Algorithm(char);//Конструктор з параметрами

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

bool LoadData(char); virtual void Run(); // Метод для запуску алгоритма

int GetStrCount(std::ifstream&); // Зчитуємо кількість строк у файлі

int GetColCount(std::ifstream&); // Зчитуємо кількість стовпців у файлі

virtual bool validateData(); // Метод для перевірки значень у матриці. Викликається перед Run()

Клас LittleAlgorithm успадковується от Algorithm

Властивості/поля:

vector<pair<int,int>> result; // Результат.

Тут буде зберігатися шуканий цикл enum check{Row, Col};

Методи:

LittleAlgorithm();//Конструктор

LittleAlgorithm(vector<vector<int>>);//Конструктор із параметрами

LittleAlgorithm(char*);//Конструктор із параметрами

virtual void Run(); // Перевизначення метода успадкованого класу

int getMin(vector<vector<int>>, int, check); // Пошук мінімального елемента стовпця/рядка

void matrixProcedure(vector<vector<int>>); // Метод у якому йде пошук циклу

void showMatrix(vector<vector<int>>); // Вивід матриці

int getResultSum(); // Зчитування суми усіх обраних дуг

virtual bool validateData(); // Перевизначення метода успадкованого класа.

Клас MyEdit віджета Текстове поле

explicit MyEdit(QWidget *parent = nullptr);//Конструктор із параметрами

void print(QString result) //Метод друку , який визначен як слотк

Клас table Віджета Таблиця

explicit table(QWidget *parent = nullptr);//Конструктор із параметрами

void Write(QString result);//Метод який визначен як сигнал та пов’язан із слотом Текстового поля

void ChangeSize(int index)//Метод, який визначен як слот та пов’язан із віджетом Лічильник та змінює розмір таблиці

vector<pair<int,int>> Process()//Метод, визначений як слот та пов’язаний із віджетом Кнопка “Обработать”.

3.6 Програмна модель системи оптимізації транспортних витрат

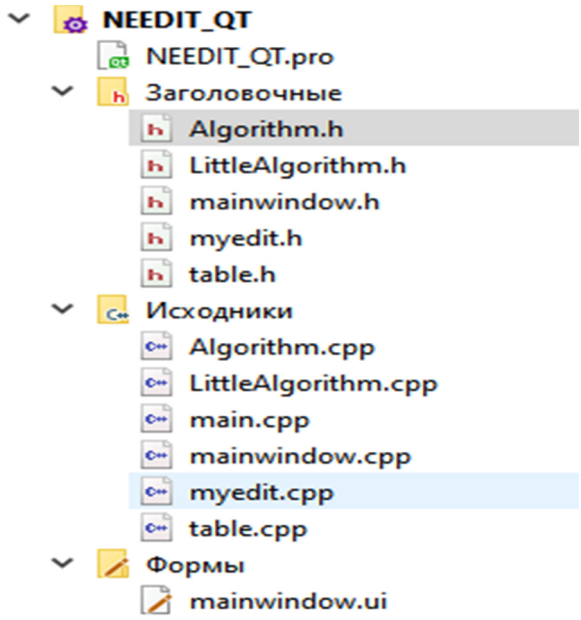


Рисунок 3.9 Програмна модель системи оптимізації транспортних витрат

Программа складається з:

Файла проекта NEEEDIT_QT.pro

Він містить відомості про проект, список заголовних файлів, список файлів Вихідного коду, список файлів Форм.

Заголовні файли:

Algorithm.h:

Містить оголошення однойменного класу, оголошення і визначення його методів і властивостей.

LittleAlgorithm.h:

Містить оголошення однойменного класу, оголошення і визначення його методів і властивостей

mainwindow.h:

Містить оголошення класу головного вікна.

myedit.h:

Містить оголошення класу віджета Текстового поля розміщеного на формі, визначення методів і властивостей, цього класу.

table.h:

Містить оголошення класу віджета Таблиці розміщеної на формі, визначення методів і властивостей, цього класу.

Вихідні файли:

Algorithm.cpp :

Визначення і перевизначення методів класу

Algorithm LittleAlgorithm.cpp :

Визначення і перевизначення методів класу LittleAlgorithm

main.cpp :

Файл містить точку входу, ініціалізація класу Головного вікна

mainwindow.cpp:

Перевизначення конструктора і деструктора для класу Головного вікна

myedit.cpp :

Перевизначення класу віджета Текстового поля

table.cpp:

Перевизначення класу віджета Таблиця

Файл формы:

mainwindow.ui :

Визначає вид Головного вікна, розташування і вид віджетів розміщених на головному вікні. Містить інформацію про підключених сигналах і слотах для віджетів.

3.7 Інтерфейс системи оптимізації транспортних витрат

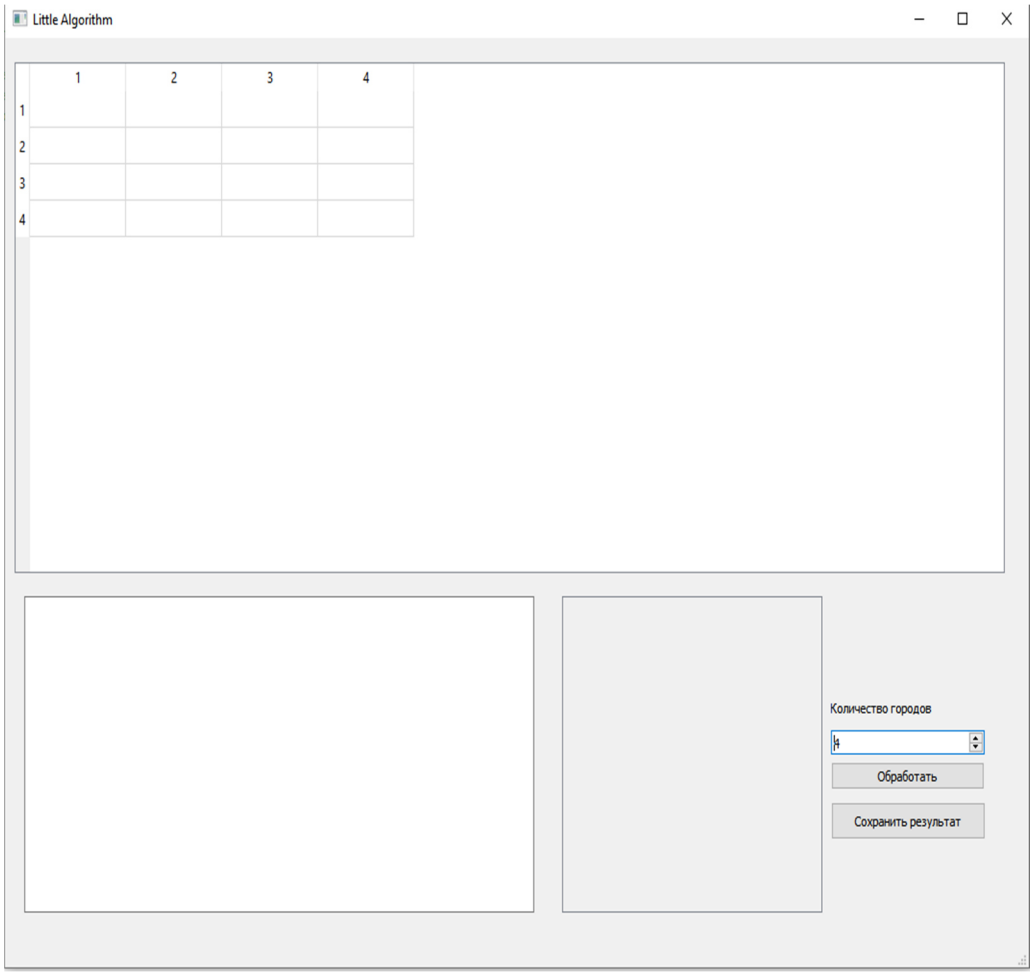


Рис. 3.9 Інтерфейс системи оптимізації транспортних витрат

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.

Опис автоматично формованої документації:

У програмі передбачена функція збереження результатів для подальшої роздруківки.

За підсумком роботи програми були отримані і оброблені наступні маршрутні листи:

«Лист», «Лист 1», «Лист 2», Лист 3», «Лист 4» - рисунки 3.10-3.14

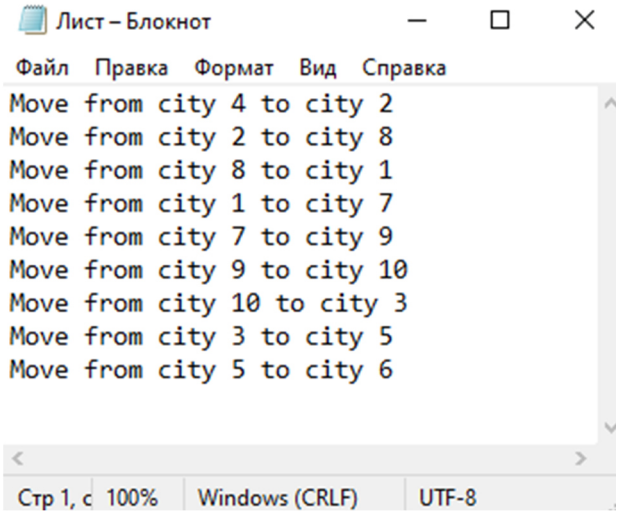


Рис. 3.10 Лист

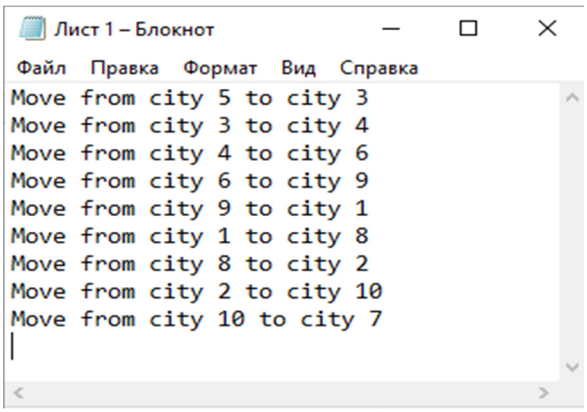


Рис. 3.11 Лист 1

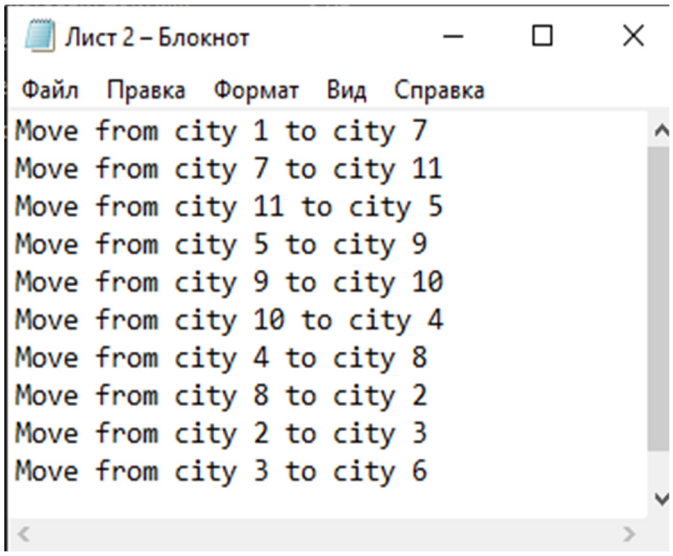


Рис. 3.12 Лист 2

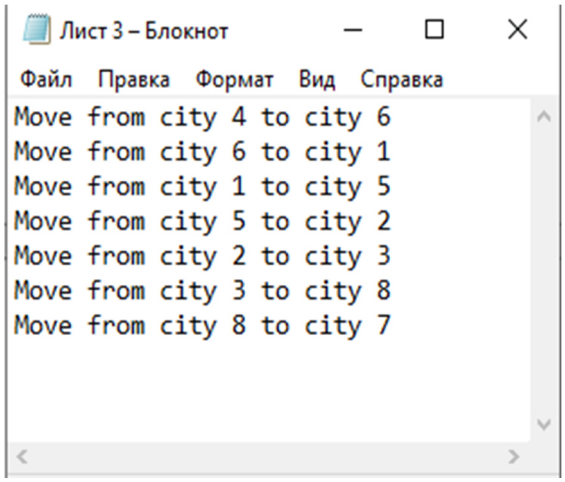


Рис. 3.13 Лист 3

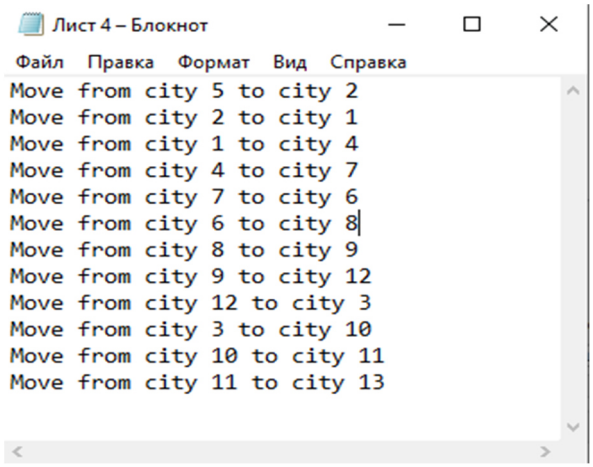


Рис. 3.14 Лист 4

Вихідні дані.

У якості вихідних значень для роботи програми використовують попередньо складені матриці відстаней.

Дані адреси згідно з проектом були розподілені на частини за районами, які у подальшому будуть обслуговуватися бригадами.

За допомогою сервісу google maps були оцінені відстані між точками маршрутів та складені матриці відстаней. Адреси та матриці відстаней зображено на таблицях 3.2 - 3.11

Таблиця 3.2

Адреси точок об'їзду для першої бригади у Кальміуському районі

№	Заклад	Адреса	буд.	місце	кількість блоків
1	ЗОШ №20	вул. Ливарна	4	1 поверх (коридор)	2
2	ЗОШ №21	вул. Озеро Хасан	65	корпус №1 (коридор) ; корпус №2 (коридор)	2
3	ЗОШ №24	проїзд Шопена	4	І поверх (хол)	2
4	ЗОШ №44	вул. Заозерна	266	Коридор	2
5	ЗОШ №45	вул. Войкова	31-а	Коридор	2
6	ЗОШ №50	вул. Невська	91	Кабінет директора	2
7	Технологічний ліцей міськради	пр. Нікопольський	133	Вестибюль	1
8	Школа №71	вул. Гонди	105А	Коридор	1
9	ЗОШ-інтернат №2	провулок Космічний	1	Навчальний корпус – в коридорі, спальний корпус – відповідальний кабінет	1
10	Ясла-садок №57	вул. Героїчна	8/10	Кабінет завідувача	1

Таблиця 3.3

Матриця відстаней точок для таблиці 3.2

	1	2	3	4	5	6	7	8	9	10
1	M	2	5	4	6	10	1	3	6	5
2	2	M	4	3	3	6	3	5	4	3
3	5	4	M	5	2	5	6	8	2	1
4	4	3	5	M	5	8	5	7	5	4
5	6	3	2	5	M	3	5	8	3	2
6	10	6	5	6	3	M	9	12	6	5
7	1	3	6	5	5	9	M	3	7	6
8	3	5	8	7	8	12	3	M	8	7
9	6	4	2	5	3	6	7	8	M	2
10	5	3	1	4	5	5	6	7	2	M

Таблиця 3.4

Адреси точок об’їзду для другої бригади у Жовтневому 1 районі

№	Заклад	Адреса	буд.	місце	кількість блоків
1	Ясла-садок №83	пр. Миру	84а	Кабінет завідувача	1
2	Ясла-садок №104	вул. Зелинського	43	Кабінет завідувача	1
3	Ясла-садок №106	вул. Куприна	25а	Кабінет завідувача	1
4	Ясла-садок №108	б-р Шевченка	246	Кабінет завідувача	1
5	Ясла-садок №120	бульвар Шевченка	256	Кабінет завідувача	1
6	Ясла-садок №125	пр.Будівельників	102	Кабінет завідувача	1
7	Ясла-садок №129	вул. Котляревського	10	Кабінет завідувача	1
8	Ясла-садок №130	пер. Трамвайний	12	Кабінет завідувача	1
9	Ясла-садок №134	вул. Троїцька	85	Кабінет завідувача	1
10	Ясла-садок №135	вул.Троїцька	83	Кабінет завідувача	1
11	Ясла-садок №136	вул.П.Орлика	94а	Кабінет завідувача	1

Таблиця 3.5

Матриця відстаней для таблиці 3.4

	1	2	3	4	5	6	7	8	9	10	11
1	м	2	3	4	3	1	3	2	3	3	4
2	2	м	1	2	3	1	5	1	2	2	3
3	3	1	м	2	2	1	5	1	2	2	3
4	4	2	2	м	1	3	4	2	1	1	2
5	3	3	2	1	м	3	4	2	1	1	1
6	1	1	1	3	3	м	4	1	2	3	4
7	3	5	5	4	4	4	м	4	4	4	4
8	2	1	1	2	2	2	4	м	2	2	3
9	3	2	2	1	1	1	4	2	м	1	2
10	3	2	2	1	1	1	4	2	1	м	2
11	4	3	3	2	1	1	4	3	2	2	м

Таблиця 3.6

Адреси точок об'їзду третьої бригади у Жовтневому 3 районі

№	Заклад	адрес	буд.	місце	кількість
1	Ясла-садок №139	б-р Шевченка	295а	Кабінет завідувача	1
2	Ясла-садок №146	вул. Митрополитська	96	Кабінет завідувача	1
3	Ясла-садок №148	вул. Михайла Грушевського	17	Кабінет завідувача	1
4	Ясла-садок №150	вул. Казанцева	31-а	Кабінет завідувача	1
5	Ясла-садок №156	б-р Шевченка	337	Кабінет завідувача	1
6	Ясла-садок №157	вул. Бахмутська	124	Кабінет ст.медсестри	1
7	Ясла-садок №163	бульвар Шевченка	68а	Кабінет завідувача	1
8	Ясла-садок №164	пр. Металургів	125а	Кабінет завідувача	1

Таблиця 3.7

Матриця відстаней для таблиці 3.6

	1	2	3	4	5	6	7	8
1	M	2	1	3	1	1	2	3
2	2	M	2	4	2	3	3	4
3	1	2	M	4	1	2	4	4
4	3	4	4	M	4	3	1	1
5	1	2	4	4	M	2	3	4
6	1	3	3	3	2	M	2	3
7	2	3	1	1	3	2	M	1
8	3	4	1	1	4	3	1	M

Таблиця 3.8

Адреси точок об'їзду четвертої бригади у Жовтневому 2 районі

№	Заклад	Адреса	буд.	місце	кількість блоків
1	ЗОШ №20	вул. Ливарна	4	1 поверх (коридор)	2
2	ЗОШ №21	вул. Озеро Хасан	65	корпус №1 (коридор) ; корпус №2 (коридор)	2
3	ЗОШ №24	проїзд Шопена	4	І поверх (хол)	2
4	ЗОШ №44	вул. Заозерна	266	Коридор	2
5	ЗОШ №45	вул. Войкова	31-а	Коридор	2
6	ЗОШ №50	вул. Невська	91	Кабінет директора	2
7	Технологічний ліцей міськради	пр. Нікопольський	133	Вестибюль	1
8	Школа №71	вул. Гонди	105А	Коридор	1
9	ЗОШ-інтернат №2	провулок Космічний	1	Навчальний корпус – в коридорі, спальний корпус – відповідальний кабінет	1
10	Ясла-садок №57	вул. Героїчна	8/10	Кабінет завідувача	1

Таблиця 3.9

Матриця відстаней для таблиці 3.8

	1	2	3	4	5	6	7	8	9	10
1	M	2	5	4	6	10	1	3	6	5
2	2	M	4	3	3	6	3	5	4	3
3	5	4	M	5	2	5	6	8	2	1
4	4	3	5	M	5	8	5	7	5	4
5	6	3	2	5	M	3	5	8	3	2
6	10	6	5	6	3	M	9	12	6	5
7	1	3	6	5	5	9	M	3	7	6
8	3	5	8	7	8	12	3	M	8	7
9	6	4	2	5	3	6	7	8	M	2
10	5	3	1	4	5	5	6	7	2	M

Таблиця 3.10

Адреси точок для п’ятої бригади у Східному районі

№	Заклад	Адреса	буд.	місце	кількість блоків
1	Спеціалізована школа №5	вул. Київська	72	Вестибюль (1 поверх)	1
2	ЗОШ №41	вул. Олімпійська	67	Кабінет директора	1
3	ЗОШ №56	Морський бульвар	8	Коридор	1
4	ЗОШ №61	вул. Волгоградська	1	Кабінет № 1.1	1
5	Школа-ліцей №69	пр. М.Жукова	92	Вестибюль	1
6	Ясла-садок №35	вул. Пашковського	13	Кабінет завідувача	1
7	Ясла-садок №46	вул. Московська	36-а	Кабінет завідувача	1
8	Ясла-садок №56	пр.Перемоги	35-А	Кабінет завідувача	1
9	Ясла-садок №67	вул. Пашковського	25а	Кабінет завідувача	1
№	Заклад	Адреса	буд.	Місце	кількість блоків
10	Ясла-садок №70	вул. Гугеля	9	Кабінет завідувача	1

11	Ясла-садок №91	вул.Ломізова	7	Спортивна зала закладу	1
12	Ясла-садок № 103	бульвар Морський	9	Кабінет завідувача	1
13	Ясла-садок №114	вул. Волгоградська	6	Кабінет завідувача	1

Таблиця 3.11

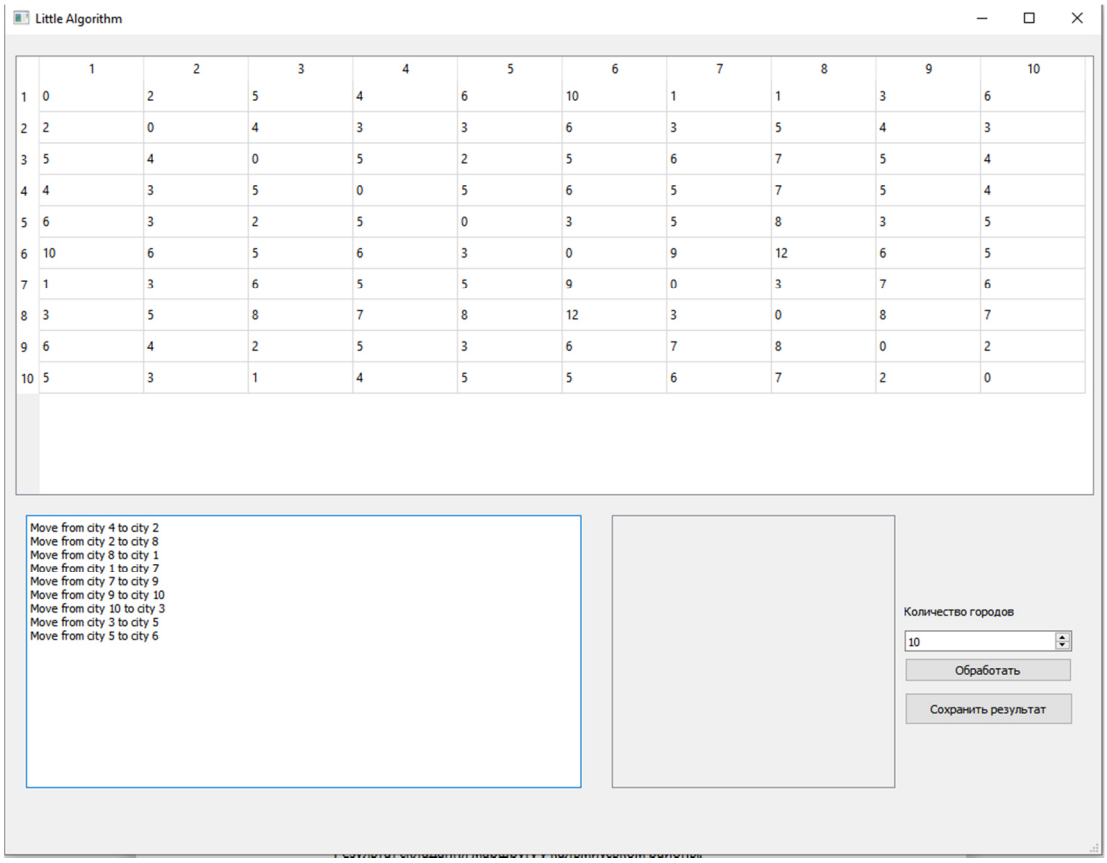
Матриця відстаней для таблиці 3.10

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	M	3	5	1	8	5	4	7	6	7	8	8	6
2	3	M	5	3	4	4	3	4	3	4	5	5	3
3	5	5	M	3	9	3	3	2	3	2	1	1	3
4	1	3	3	M	6	2	1	2	1	3	3	2	2
5	8	4	9	6	M	7	6	7	6	7	8	9	7
6	5	4	3	2	7	M	1	1	1	2	3	2	2
7	4	3	3	1	6	1	M	1	1	2	2	2	2
8	7	4	2	2	7	1	1	M	1	2	2	2	2
9	6	3	3	1	6	1	1	1	M	2	3	2	2
10	7	4	2	3	7	2	2	2	2	M	1	2	3
11	8	5	1	3	8	3	2	2	3	1	M	1	2
12	8	5	1	2	9	2	2	2	2	2	1	M	2
13	6	3	3	2	7	2	2	2	2	3	2	2	M

3.8 Результати моделювання

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 «Обробити». 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 малюнках 3.15 – 3.19.



	1	2	3	4	5	6	7	8	9	10
1	0	2	5	4	6	10	1	1	3	6
2	2	0	4	3	3	6	3	5	4	3
3	5	4	0	5	2	5	6	7	5	4
4	4	3	5	0	5	6	5	7	5	4
5	6	3	2	5	0	3	5	8	3	5
6	10	6	5	6	3	0	9	12	6	5
7	1	3	6	5	5	9	0	3	7	6
8	3	5	8	7	8	12	3	0	8	7
9	6	4	2	5	3	6	7	8	0	2
10	5	3	1	4	5	5	6	7	2	0

Рис. 3.15 Результат складання маршруту у Кальміуському районі

Інтерпретування результатів автоматично формованої документації наступне. Водій повинен відвідати адреси у наступній черзі:

1)Вулиця заозерна 266

2)Вулиця Озера Хасан 65

3)Вулиця Гонди 105а

4)Вулиця Ливарна 4

5)Проспект Бойка 133

6)Провулок Космічний 1

7)Вулиця Героїчна 8

8)Прїзд Шопена 4

9)Вулиця Войкова 31а

10)Вулиця Невська 91

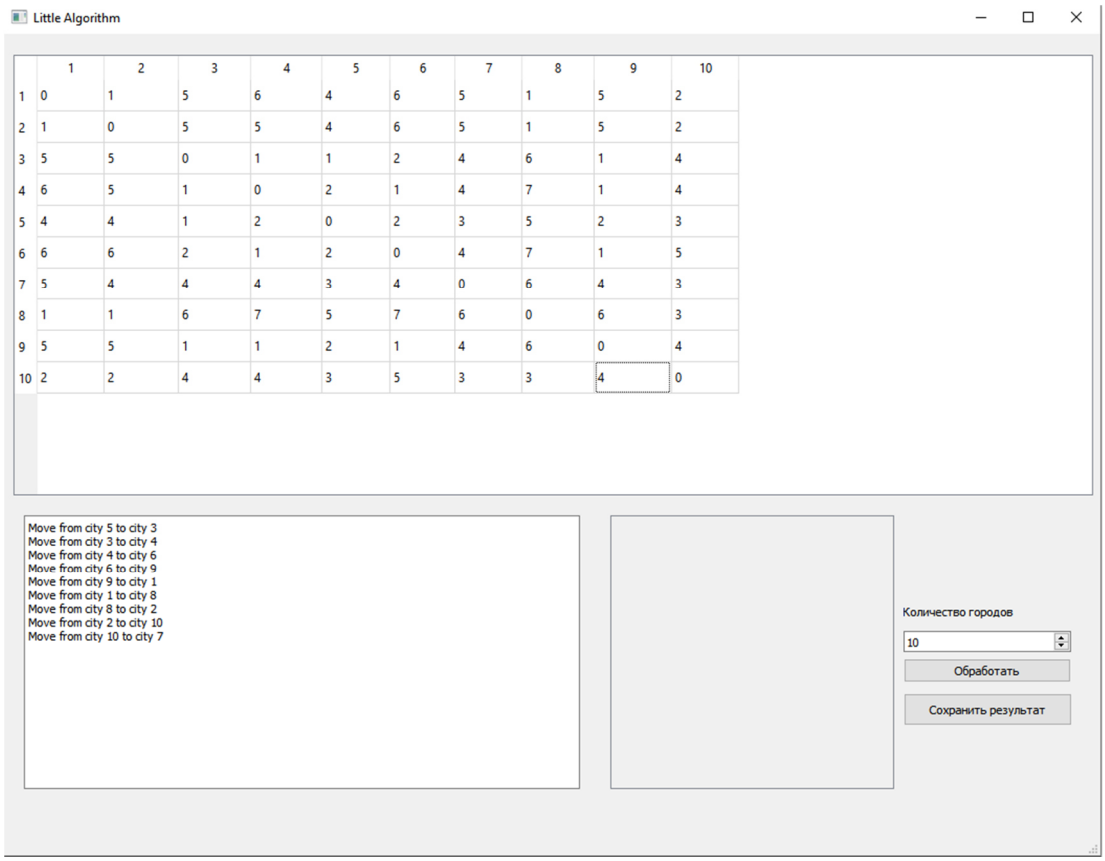


Рис.3.16 - Результат складання маршруту у Жовтневому 1 районі

Інтерпретування результатів автоматично формованої документації наступне. Водій повинен відвідати адреси у наступній черзі:

1)Вулиця Казанцева 27а

2)Проспект Мира 79

3)Вулиця Пушкіна 51

4)Вулиця Семенішина 46

5)Вулиця Італійська 116а

6)Вулиця Пилипа Орлика 90а

7)Вулиця Гранітна 116а

8) Вулиця пилипа Орлика 92

9) Вулиця Таврійська 25а

10)Вулиця Грецька 206

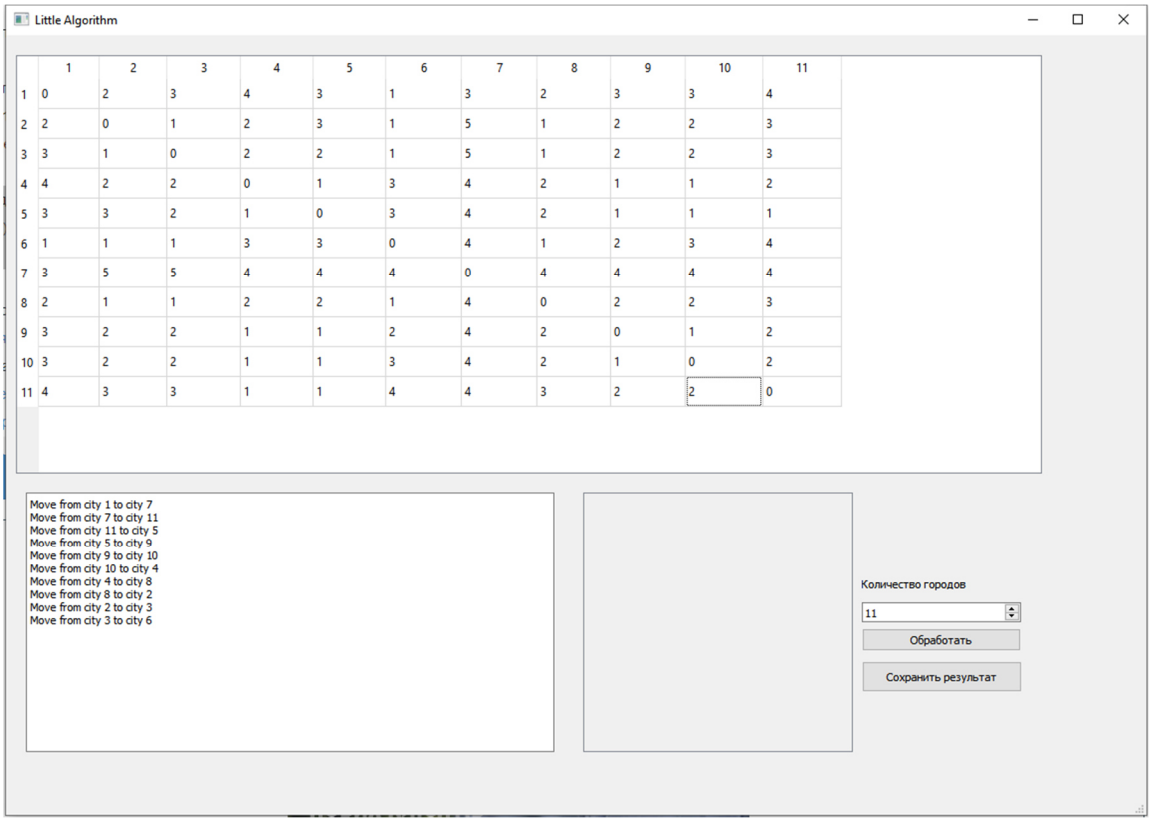


Рис 3.17 - Результат складання маршруту у Жовтневому 2 районі

Інтерпретування результатів автоматично формованої документації наступне. Водій повинен відвідати адреси у наступній черзі:

1)Проспект Миру 84а

2)Вулиця Котляревського 10

3)Вулиця Пилипа Орлика 94а

4)Бульвар Шевченка 256

5)Вулиця Троїцька 85

6)Вулиця Троїцька 83

7)Бульвар Шевченка 246

8)Провулок Трмвайний 12

9)Вулиця Зелінського 43

10)Вулиця Купріна 25а

11)Проспект Будівельників 102

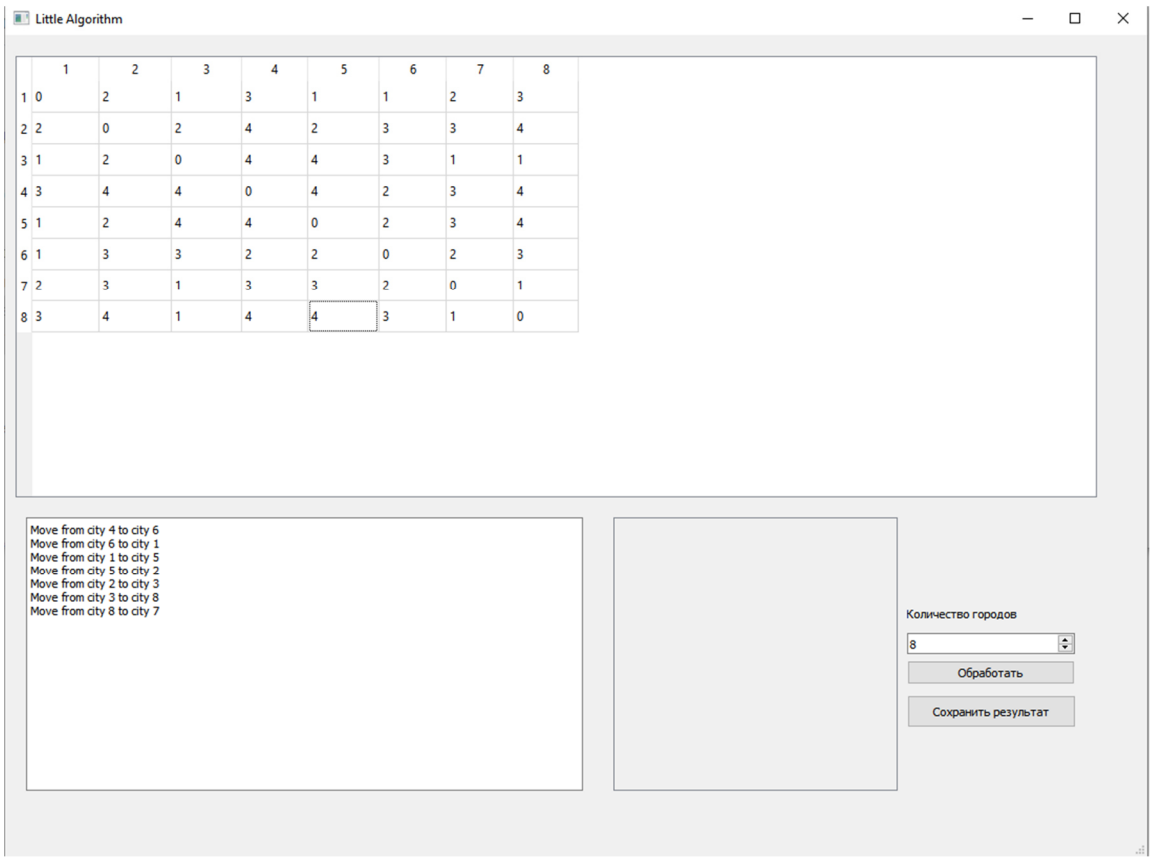


Рис. 3.18 - Результат складання маршруту у Жовтневому 3 районі.

Інтерпретування результатів автоматично формованої документації наступне. Водій повинен відвідати адреси у наступній черзі:

1)Вулиця Казанцева 31а

2)Вулиця Бахмутська 124

3)Бульвар Шевченка 295а

4)Бульвар Шевченка 337

5)Вулиця Митрополитська 96

6)Вулиця Михайла Грушевського 17

7)Вулиця Металургів 125а

8)Бульвар Шевченка 68а

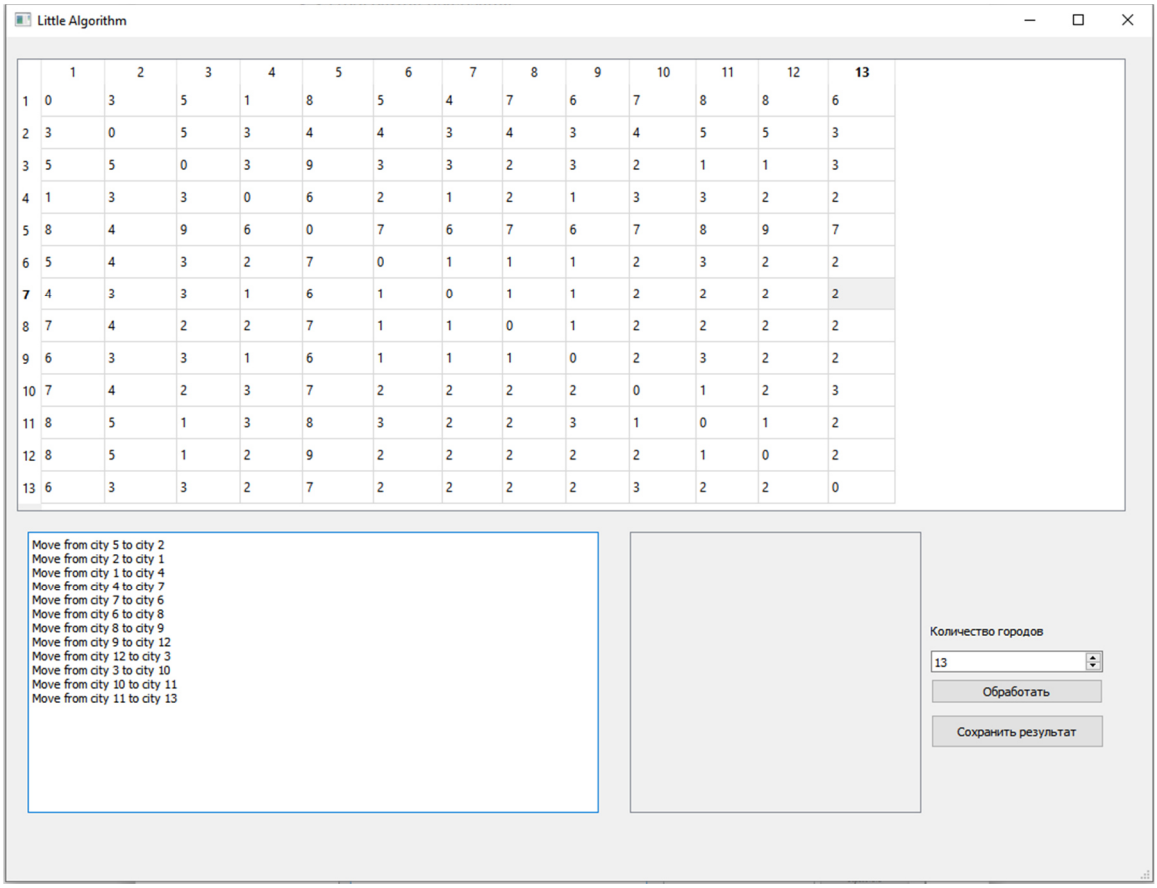


Рис. 3.19 - Результат складаня маршруту у Східному районі.

Інтерпретування результатів автоматично формованої документації наступне. Водій повинен відвідати адреси у наступній черзі:

1)Проспект М. Жукова 92

2)Вулиця Олімпійська 67

3)Вулиця Київська 72

4)Вулиця Волгоградська 1

5)Вулиця Московська 36а

6)Вулиця Пашковського 13

7)Проспект Перемоги 35а

8)Вулиця Пашковського 25а

9)Бульвар Морский 9

10)Бульвар Морський 8

11)Вулиця Гугеля 9

12)Вулиця Ломізова 7

13)Вулиця Волгоградська 6

Таким чином були складені та отримані маршрутні листи для руху бригад в умовах міста Маріуполя.

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі кваліфікаційної роботи була наведена характеристика підприємства, проаналізована діяльність його відділів, поставлене завдання.

Для вирішення наведеного завдання були складені матриці відстаней та успішно розроблена система оптимізації транспортних витрат та отримані оптимальні маршрути відвідування адрес. Дана система може бути використана для інших проектів та задач, які потребують складання оптимальних маршрутів.

Точний метод гілок та меж підтвердив, що є одним із успішних методів для вирішення задач складання оптимальних маршрутів за наявності невеликої кількості точок. Також цей метод є одним з найкращих методів вирішення асиметричних завдань комівояжера.

Розробка системи та її подальше використання на підприємстві впливатиме на підприємство позитивно. Завдяки оптимізації маршрутів можна не тільки скоротити витрати на пальне та зменшити час виконання проектів, але й підвищити ефективність роботи.

ВИСНОВКИ

У висновку до кваліфікаційної роботи відзначимо, що автотранспортні витрати впливають на вартість продукції та послуг, що надає підприємство.

Якщо робочий процес автотранспортних засобів підприємства оптимізован таким чином, щоб вчасно задовольнити потреби клієнтів та максимально скоротити витрати, то підприємство має змогу пропонувати більш привабливі ціні на свою продукцію та послуги на відміну від конкурентів. Тому оптимізація транспортних витрат є однією із головних проблем підприємств, що пропонують послуги, які будь-якою мірою пов’язані із транспортуванням.

У ході виконання кваліфікаційної роботи була досягнута її мета та виконані усі поставлені завдання:

- проаналізовано існуючі підходи та методи комбінаторної оптимізації для оптимізації транспортних витрат;
- сформульоване завдання комівояжера та наведено класифікацію його видів;
- проаналізовано існуючі методи та алгоритми вирішення проблеми комівояжера;
- проаналізовано роботу ІТ-підприємства «Інтегровані системи безпеки»;
- побудовано систему оптимізації автотранспортних витрат ІТ-підприємства «Інтегровані системи безпеки».

У першому розділі кваліфікаційної роботи були розглянуті комбінаторна оптимізація та її методи для скорочення транспортних витрат, сформульоване та розглянуте завдання комівояжера та класифікація його різновидів.

У другому розділі кваліфікаційної роботи були проаналізовані моделі та методи вирішення завдання комівояжера та обрано метод для подальшої реалізації.

У третьому розділі кваліфікаційної роботи була наведена характеристика підприємства, проаналізована діяльність його відділів, поставлене завдання, складені матриці відстаней та успішно розроблена система оптимізації транспортних витрат, отримані оптимальні маршрути відвідування адрес.

Завдяки оптимізації маршрутів можна не тільки скоротити витрати на пальне та зменшити час виконання проектів, але й підвищити ефективність роботи.

Таким чином була побудована система оптимізації транспортних витрат для підприємства «Інтегровані системи безпеки».

Літературні джерела

1. Костевич Л. С. Математическое программирование: Информ. технологии оптимальных решений: Учеб. пособие / Л. С. Костевич. — Мн.: Новое знание, 2003. ил., стр. 150

2. Поборчий І. В./ Дослідження евристичних методві вирішення завдання комівояжера/ І. В. Поборчий - 2016. – 44с

3. Land, A. H. et al.,. An automatic method of solving discrete programming problems (англ.) // Econometrica : журнал. — 1960. — Vol. 28, no. 3. — P. 497—520.

4. Білоусов А.І., Ткачов С. Б./ Дискретна математика. / Булоусов А.і., Ткачов С. Б. — м.: МГТУ, 2006. - 744 с

5. Кормен, Т., Лейзерсон, Ч., Рівест, Р., Штайн, К. Глава 23. Мінімальні остовні дерева / / алгоритми: побудова і аналіз = Introduction to Algorithms / Под ред. І.в. Красикова. - 2-е вид. - М.: Вільямс, 2005. - 1296 с

6.Левітін А. В./ Алгоритми. Введення в розробку і аналіз — М.: Вільямс, 2006. - С. 159-160. - 576 с

7. Берж К. "Теорія графів та її застосування", м., ІЛ, 2008;

8. Гончаров, Е. Н. Дослідження операцій. Приклади і завдання Навчальний посібник / Е. Н. Гончаров, А. І. Єрзін, В. В. Залюбовський. 2005. – 78 с.

9. С.А. Ашманов, А.В. Тимохов. Теориія оптимизації в задачах та вправах. – СПб.: Лань, 2012. – 448 с.

10. А. В. Кузнєцов, Н. І. Холод, Л. С. Костевич. Керівництво до вирішення задач з математичного програмування. — Мінськ: Вища школа, 1978. — С. 110.

11. В. І. Мудров. Задача про комівояжера. - М. «Знання», 1969. - С. 62.

12. Костюк Ю. Л. Ефективна реалізація алгоритму розв'язання задачі комівояжера методом гілок і меж // Прикладна Дискретна математика. Обчислювальні методи в дискретній математиці, 2010. 41 №2 (20). С. 78-90.

13. Parragh S., Doerner K., Hartl R. A survey on pickup and delivery problems. Part II: Transportations between customers and depot // J. Betriebswirtschaft. 2008. V. 58. No 2. P. 81-117.

14. Андерсон, Джеймс. Дискретна математика і комбінаторика = Discrete Mathematics with Combinatorics. - М.: «Вільямс", 2006. - С. 960

15. Липський в. комбінаторика для програміста. - М.: Мир, 1988. - 213 с.

16. Райгородський А.М. лінійно-алгебраїчні та імовірнісні методи в комбінаториці. - Літня школа «Сучасна математика». - Дубна, 2006.

17. Рейнгольд Е., Нівергельт Ю., Део Н. комбінаторні алгоритми. Теорія і практика. - М.: Мир, 1980. - 476 с.

18.Томас х. Кормен, Чарльз і. Лейзерсон, Рональд л.Рівест, Кліффорд Штайн алгоритми: побудова іаналіз = Introduction to Algorithms. - 2е Вид. - М.: «Вільямс", 2006. - С. 1296

19. Корбут А. а., Фінкельштейн Ю. Ю. дискретне програмування. - М.: Наука, 1969. - 368 с

20. Міну м. Математичне програмування. Теорія і алгоритми. - М.: Наука, 1990. - 488 с.

21. 1. Беллман, р. динамічне програмування-М.: ІЛ, 1960. - 400 С.

22. Беллман, р. прикладні завдання динамічного програмування-М.: Наука, 1965. - 457 с.

23. Сігал і. Х., Іванова А.П. введення в прикладне дискретне програмування: моделі та обчислювальні алгоритми. М.: ФИЗМАТЛИТ, 2003. -- 240 с.

24. Р. Беллман, с. Дрейфус прикладні задачі динамічного програмування - М., 1965 р., 460 стор.

25. Галяутдінов Р.Р. Завдання комівояжера-метод гілок і кордонів // Сайт викладача економіки. [2020]. URL: http://galyautdinov.ru/post/zadacha-kommivoyazhera

26. Карнаух Т.О., Ставровський А.Б./ Теорія графів у задачах: М.Київ. КНУ ім. Тараса Шевченка. факультет кібернетики 2012р. 90 стр.

27. Костюк Ю. Л. Основи алгоритмізації. Навчальний посібник. // Вид-во ТГУ, 1996 р. - 122 с.

28. Костюк Ю.Л. Завдання комівояжера: наближений алгоритм за методом гілок і меж з гарантованою точністю --- // Прикладна Дискретна математика. Випуск № 45, 2019 р. с. 104-112

29. Костюк Ю.Л. збалансована задача до комівояжерів для несиметричних відстаней --- // Інформаційні технології та математичне моделювання (ІТММ-2016): Матеріали XV Міжнародної конференції ім. А. Ф. Терпугова (12-16 Вересня 2016 р.) - 2016. - Ч. 2. с. 43-46

30. Костюк Ю. Л. Основи програмування. Розробка та аналіз алгоритмів: Навч. посібник. - Томськ: Вид-во Том. ун-ту, 2006. - 244с.

31. Хелд м., Карп Р. М. застосування динамічного програмування до завдань упорядкування // Кібернетичний збірник. Вип. 9, Стара серія. М.: Мир, 1964. С. 202-218.

32. Данциг Дж. Лінійне програмування, його застосування та узагальнення: Пер.з англ. М.: Прогрес, 1966.

33. Меламед і. І., Сергєєв С. і., Сігал і. Х. Завдання комівояжера. Точні алгоритми / / Автоматика і телемеханіка. 1989. № 10. С. 3-29.

34. Рейнгольд Е., Нівергельт Ю., Део Н. комбінаторні алгоритми. Теорія і практика: Пер.з англ. М.: Мир, 1980. 478 с.

35. С. Ф. Подшивалов, К. С. Подшивалова, Л. В. Левицька, С. В. Дунаєв
Наукова робота «удосконалення алгоритму методу гілок і меж у задачі комівояжера для автомобільного транспортного графа»

36. Пожидаєв, м. с. алгоритми розв'язання задачі маршрутизації транспорту: дис. канд. техн. наук / Пожидаєв м.с. - Томськ, 2010. - 134 с.

37. Прокоф'єва, О. С. Розробка методики оптимізації розвізних маршрутів: дис. канд. техн. наук / Прокоф'єва О.С. - Іркутськ, 2004. - 167с.

38. Goldberg D.E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison - Wesley Publishing Company, Inc., 1989.

39. Вороновський Р. К.,Махотило К. В.,Петрашев С. А., Сергєєв С. А. Генетичні алгоритми, штучні нейронні мережі і проблеми віртуальної реальності. Харків: Основа, 1997. - 112 с.

40. Fisher R.A. The genetical theory of natural selection. -Oxford: Clarendon press, 1930. - 362 p.

41. Батищев Д. і. генетичні алгоритми вирішення екстремальних завдань. Воронеж: ВГТУ, 1995 Гавриляко в.м., Соколов А. В. чисельне рішення негладких рівнянь у задачі швидкодії // Вюник Харювського Уш, сер. Математика, прикладна математика і мехашка. - 2002. - № 542. - С. 145 - 148.

42. Самарський А. а., Михайлов А. П. Математичне моделювання: ідеї, методи, приклади.- М.: Наука, Фізматліт, 1997.-320 с.

43. Прокудін Р. С. Методи оптимізації перевезень у транспортних мережах // Проблеми інформатизації та управління: Збірник наукових праць: Випуск 13.- К.: НАУ, 2005.- З 88-97.

44. С. а. Канцедал, м. в. Костікова «динамічне програмування для задачі комівояжера»

45. Мудров В. і. завдання про комівояжера. М.: Знання, 1969. 62 С.

46.Пекарська С.С. побудова раціонального маршруту при вирішенні завдань транспортної логістики з урахуванням навантаження в дорожній мережі // збірник «перспективи розвитку інформаційних технологій» праці Всеросійської молодіжної науково-практичної конференції. Кузбаський державний технічний університет імені Т. Ф. Горбачова, Міжнародний науково-освітній центр КузГТУАгепа Multimedia. 2014. С. 382-383.

47.Седжвік р. фундаментальні алгоритми на С++. Частини 1-4. Аналіз. Структури даних. Сортування. Пошук = Algorithms in C++, СПб: Діасофт, 2002. 688 с.

48.Ураков А.Р., Михтанюк а. а. оцінка кількості варіантів обходу в задачі комівояжера з додатковими умовами //Глобальний 42 науковий потенціал, 2012. № 21. С. 82-86. 49. Хухрянська Є.С., Юдіна Н.Ю., Ющенко є. в. Аналіз можливостей застосування методів теорії розкладів до завдань

деревообробних виробництв та їх формалізація // лісотехнічний журнал. 2011. № 3. С. 37-40.

50.Bang-Jensen J., Gutin G., Yeo A. When the greedy algorithm fails // Discrete Optimization. 2004 Vol. 1, No 2. P. 121-127. 20.Berbeglia G., Cordeau J.F., Gribkovskaia I., Laporte G. Static pickup and delivery problems: A classification scheme and survey // TOP. 2007. V. 15. No 1. P. 1-31.

51.M. Dorigo, V. Maniezzo & A. Colorni. Ant System: optimization by a colony of cooperating agents // IEEE transactions on systems, man, and cyberneticspart B. 1996 No. 26 (1). P. 29-41.

52.Gutin G., Jakubowicz H., Ronen, Sh., Zverovitch A. Seismic vessel problem, Communications in DQM, 2005 No. 8. P. 13-20.

53.Hahsler M., Hornik K. TSP – Infrastructure for the traveling salesperson problem // Journal of statistical software, 2007 No. 23 (2). P. 1-21.

54.Kellerer H., Pferschy U., Pisinger D. Knapsack problems. // Springer-Verlag, 2003. 548 c.

55.Little J. D. C., Murty K. G., Sweeney D. W., Karel C. An algorithm for the traveling salesman problem // Operations Research. 1963 Vol. 11, No 6. P. 972-989.

56.Paar C., Pelzl J. Understanding cryptography: a textbook for students and practitioners. Berlin: Springer, 2010. 372 c.

57.Parragh S., Doerner K., Hartl R. A survey on pickup and delivery problems. Part II: Transportations between customers and depot // J. Betriebswirtschaft. 2008. V. 58. No 2. P. 81-117.

59.Пекарська С.С. Розрахунок вагових коефіцієнтів для знаходження найкоротшого за часом шляху // Збірник доповідей Всеросійської науково-практичної конференції «Сучасні техніка і технології», Томськ, 2012.

60. Кормен Т. Х., Лейзерсон Ч. І., Рівест Р. Р., Штайн К. Алгоритми. Побудова та аналіз. 2 изд. М.: Вільямс, 2012. 1296 с.

61. Курейчик В.В., Курейчик в. м. генетичний алгоритм визначення шляху комівояжера // Известия Російської академії наук. Теорія і системи управління. 2006. № 1. С. 94-100.

62.Курейчик в.м., Кажаров а. а. мурашині алгоритми для вирішення транспортних завдань. // Известия РАН. Теорія і системи управління. – 2010. № 1. С. 32-45.

63.Левітін а. в. алгоритми: введення в розробку і аналіз. М.: Вільямс, 2006. 576с.

64. Вишняков П.О. планування маршрутів з використанням модифікованого методу «найближчого сусіда» //математичні методи в техніці і технологіях - ММТТ. 2014. № 6 (65). С. 63-67.

65. Карандєєв Д. Ю., Голубничий А. а. реалізація методу гілок і меж в статистичному середовищі Я // інтернет-журнал «наукознавство» Том 7, №6 (2015) http://naukovedenie.ru/PDF/80TVN615.pdf

66. Міфтахова А. а. МЕТОД відсікання гілок ДЕРЕВА рішень / / Науковий журнал "бюлетень науки і практики" / / Поволзький державний університет телекомунікацій та інформації 2016 № 4

67. Мацій О.Б. підвищення точності симетричної задачі класу комівояжера великої розмірності // Вісник Харківського національного автомобільно-дорожнього університету. 2011. № 55. С. 100-102.

68. Посипкін М.А. рішення задач глобальної оптимізації в середовищі розподілених обчислень. Програмні продукти і системи, 2010, т. 1, с. 23 - 29.

69. Арі де Брюн, Олександр Х. Г., Ріннуй Кан, Гаррі У. Дж. М. Триенекенс. Інструмент моделювання для оцінки продуктивності паралельних розгалужених і пов'язаних алгоритмів. Математичне програмування, Квітень 1988, том 42, випуск 1-3, стор.

70. Євтушенко Ю., Посипкін м., Сігал і. структура паралельної великомасштабної глобальної оптимізації. Інформатика - дослідження і розробки, 2009, Том 23, випуск 3 - 4, с. 211 - 215.

71. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. Communication of the ACM, July 1978, vol. 21, issue 7, pp. 558 - 565.

72. Копилова Е. С., Ніколаєва Д. С., Бунтова Е. В. рішення задачі комівояжера з використанням методу гілок і меж – Том 4 № 4[Електронний ресурс] URL: http://progresscвободный. - Загл. З екрану яз. русий., англ.

73. Лазарєв Е.а. про застосування евристик для методу гілок і меж рішення задачі оптимізації мережі передачі даних //праці НГТУ ім. Р. є. Алексєєва / / 2012р. УДК 519.8

74. Гольштейн є.Г. завдання лінійного програмування транспортного типу /є. Г. Гольштейн, Д. Б. Юдін. М.: Наука, 1969. 384 с.

75.Кузнєцов А. В.керівництво до вирішення задач з математичного програмування: Навчальний посібник / А. В. Кузнєцов, Н. І. Холод, Л. С. Костевич. Мінськ, 2001. 448с

76. Е. А.Попов und М. є. Семенкіна. Еволюційні алгоритми моделювання та оптимізації. - М.: LAP LAMBERT Academic Publishing, 2013. - 72 С.

77. Г. І. Просвєтов. Математичні методи в логістиці. Завдання і рішення. Навчально-практичний посібник. - М.: Альфа-Прес, 2014. - 304 с.

78. Раміз Гіндуллін, Юхим Бронштейн. Методи побудови оптимальних маршрутів доставки вантажів. - М.: Palmarium Academic Publishing, 2014. - 96 с.

79. О.А. Сдвижков. Дискретна математика і математичні методи економіки із застосуванням VBA Excel. - М.: ДМК прес, 2016. - 212 с.

80. Г. П. Фомін. Економіко-математичні методи і моделі в комерційній діяльності. Підручник. - М.: Юрайт, 2015. - 464 с.

81. І. Х. Сігал, А.П. Іванова. Введення в прикладне дискретне програмування. – М.: ФИЗМАТЛИТ, 2007. - 304 с.

83. А. в. Іпатов, "модифікований метод імітації відпалу в задачі маршрутизації транспорту", Тр. ІММ УрО РАН, 17, № 4, 2011, 121-12584. В. В. Захаров, А. В. Мугайских, “Динамическая адаптация генетического алгоритма маршрутизации транспорта на больших сетях”, УБС, 73 (2018), 108–133

85. Glover F., Kochenberger G.A. Handbook of Metaheuristics. New York: Kluwer Academic Publishers, 2003. 560 p.

86. Карпенко А. П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой. М.: МГТУ им. Н. Э. Баумана, 2014. 446 с. ISBN 978-5-7038-3949-2

87. Schrijver, Alexander . Combinatorial Optimization: Polyhedra and Efficiency (англ.). — Springer. — Vol. 24. — (Algorithms and Combinatorics).

88. Александр Швец und Евгений Семенкин. Коэволюционный асимптотический генетический алгоритм. – М.: LAP Lambert Academic Publishing, 2012. – 96 с.

89. Павел Галушин. Асимптотический вероятностный генетический алгоритм. – М.: LAP Lambert Academic Publishing, 2012. – 108 с.

90. Электронный источник: https://habr.com/ru/news/t/522718 / " исследователи смогли преодолеть барьер в улучшении решения задачи коммивояжера»

91. Владислав Браништи. Исследование эффективности генетического алгоритма. – М.: LAP Lambert Academic Publishing, 2014. – 88 с.

92. Ю. Л. Костюк. Ефективна реалізація алгоритму рішення завдання комівояжера методом гілок і кордонів, 2013. – 78-90 с.

93. Дискретна оптимізація та моделювання в умовах невизначеності даних Перепелиця В. А., Тебуєва Ф. б. Видавництво: Академія природознавства Рік видання: 2007 ISBN: 978-5-91327-013-9

94. Алгоритм Прима. URL: https://algowiki-project.org/ru/Алгоритм_Прима

95. Задача комівояжера. URL: https://uk.wikipedia.org/wiki/Задача_комівояжера

96. Алгоритм Крускала. URL: https://algowiki-project.org/ru/Алгоритм_Крускала

97. «Дослідження евристичних методів вирішення Завдання комівояжера».URL: https://nauchkor.ru/uploads/documents/587d365c5f1be77c40d58da6.pdf.

98. QT.UL: https://ru.wikipedia.org/wiki/Qt

99. Nelder J.A., Mead R., A Simplex Method For Functin Mnimization, Computer J., No 7, 1964 P. 308-313.

100. Qt Creator – URL: https://soft.mydiv.net/win/download-Qt-Creator

Genetic Algorithms in Search Optimizations and Machine Learning. – Addison Wesly, 1989.

101. Кормен Т. Х., Лейзерсон Ч. І., Рівест Р. Р., Штайн К. Алгоритми. Побудова та аналіз. 2 изд. М.: Вільямс, 2012. 1296 с.

102. Піскорська О. В. Оптимізація транспортних витрат методом гілок та меж. Дебют: збірник доповідей студентів. Маріупольский державний університет, економіко-правовий факультет. 20.03.2020, 310-311с.

Додаток А

Листинг

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 текущую 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){;}
```