МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій
Кафедра комп'ютерних наук та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Інформаційно-довідкова система для дослідження і аналізу ефективності інженерних мереж підприємства

Рівень вищої освіти _____другий (магістерський)_____
(перший (бакалаврський) / другий (магістерський))

Спеціальність _____122 Комп'ютерні науки _____
(код і найменування спеціальності)

Освітня програма _____комп'ютерні науки

Виконав: студент групи__МгІТ-1-22____

_____Нирко М. В._____
(прізвище та ініціали)

Науковий керівник к.т.н., доц. Яхно В. М.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент___д.т.н., проф.Чупринка В.І.

Київ 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет / Інститут ___МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ_____
(повна назва факультету/інституту)

Кафедра ____КОМП'ЮТЕРНИХ НАУК_____
(повна назва кафедри)

Рівень вищої освіти _____другий (магістерський)_____
(перший (бакалаврський) / другий (магістерський))

Спеціальність _____122 Комп'ютерні науки_____
(код і назва спеціальності)

Освітня програма _____комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри ___КН____
(абревіатура кафедри)
_____ В.Ю. Щербань
(підпис)
«_____» _____ 2023р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЄКТ) СТУДЕНТУ

Нирку Миколі Володимировичу

1. Тема кваліфікаційної роботи *Інформаційно-довідкова система для дослідження і аналізу ефективності інженерних мереж підприємства*, Науковий керівник роботи Яхно Володимир Михайлович, к.т.н., доцент, затверджені наказом КНУТД від " 12 " вересня 2023 року № 210 -уч

2. Вихідні дані до кваліфікаційної роботи Розробки кафедри комп'ютерних наук. Засоби проектування та побудови обчислювальних алгоритмів оптимізації. Методи проектування та побудови інформаційно-довідкових систем.

3. Зміст кваліфікаційної роботи (перелік питань, які потрібно опрацювати): Розділ 1. Постановка задачі і методи дослідження; Розділ 2. Обґрунтування моделей та методів; Розділ 3. Алгоритмічне та програмне забезпечення.

4. Дата видачі завдання _____

5. Консультанти розділів кваліфікаційної роботи роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Вступ	Яхно В.М., доц.		
Розділ 1	Яхно В.М., доц.		
Розділ 2	Яхно В.М., доц.		
Розділ 3	Яхно В.М., доц.		
Висновки	Яхно В.М., доц.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапу кваліфікаційної роботи (проєкту)	Орієнтовний термін виконання	Примітка про виконання
1	Вступ	04.09.2023	
2	Розділ 1 Постановка задачі і методи дослідження	18.09.2023	
3	Розділ 2 Обґрунтування моделей та методів	02.10.2023	
4	Розділ 3 Алгоритмічне та програмне забезпечення	16.10.2023	
5	Висновки	23.10.2023	
6	Оформлення (чистовий варіант)	30.10.2023	
7	Подача кваліфікаційної роботи (проєкту) науковому керівнику для відгуку (за 14 днів дозахисту)	13.11.2023	
8	Подача кваліфікаційної роботи (проєкту) для рецензування (за 12 днів дозахисту)	15.11.2023	
9	Перевірка кваліфікаційної роботи (проєкту) на наявність ознак плагіату (за 10 днів до захисту)	17.11.2023	
10	Подання кваліфікаційної роботи (проєкту) на завідувачу кафедри (за 7днів до захисту)		

З завданням ознайомлений:

Студент _____ _____
(підпис) (Власне ім’я та ПРІЗВИЩЕ)

Науковий керівник _____ _____
(підпис) (Власне ім’я та ПРІЗВИЩЕ)

.

АНОТАЦІЯ

Нирко М.В. Інформаційно-довідкова система для дослідження і аналізу ефективності інженерних мереж підприємства.

Дипломна магістерська робота за спеціальністю 122 - «Комп’ютерні науки». – Київський національний університет технологій та дизайну, Київ, 2023 рік.

Вивчення різних виробничих процесів можна провести за допомогою моделей, які відображають переміщення матеріалів та ресурсів між точками обробки, відтворення та використання. У дипломній роботі розглядаються проблеми, пов'язані з розробкою та аналізом таких моделей, які відтворюють завдання з переміщення матеріалів та ресурсів між місцями обробки, відтворення та використання. Дослідження здійснюється на прикладі мереж внутрішнього водопостачання. Використання програмного засобу дозволяє здійснювати обґрунтовані рішення щодо ефективності запропонованих керівних вирішень. Інформаційно-довідкова система з базовими функціями пошуку та оновлення інформації про обладнання та потоки ресурсів є фундаментом для створення інформаційних моделей та прийняття рішень.

Програмна архітектура системи реалізована з використанням стандартних програмних компонентів на максимальному рівні.

Ключові слова: дослідження операцій, керування запасами, моделі даних, інформаційно-довідкова система.

SUMMARY

Nyrko M.V. Information and reference system for research and analysis of the efficiency of engineering networks of the enterprise.

Master's thesis in specialty 122 - "Computer Science". - Kyiv National University of Technology and Design, Kyiv, 2023.

The study of various production processes can be conducted using models that depict the movement of materials and resources between processing points, replication, and utilization. The thesis addresses issues related to the development and analysis of such models that replicate tasks involving the movement of materials and resources between processing locations, replication, and utilization. The investigation is carried out using the example of internal water supply networks. The use of software enables informed decisions regarding the efficiency of proposed management solutions. An information-reference system with basic search and update functions for equipment and resource flow information forms the basis for constructing information models and decision-making.

The software architecture of the system is implemented with the maximum utilization of standard software components.

Keywords: operations research, inventory management, data models, information and reference system.

ЗМІСТ

ВСТУП

Актуальність теми полягає в тому, що оновлення та контроль за використанням внутрішніх інженерних мереж є необхідними елементами виробничого процесу. Без їхньої повноцінної реалізації неможливе успішне функціонування інших аспектів управління, таких як планування, зміни технологій та модернізація. Планування оновлень повинно постійно враховувати реальні можливості та зміни в умовах функціонування та розвитку інженерних мереж підприємства. Контроль і прогноз дієздатності спрямовані на забезпечення точної оцінки реальної виробничої ситуації, що, в свою чергу, надає можливість внести обґрунтовані корективи у заплановані кроки по оновленню та заміні інженерних мереж. Моделювання і контроль оновлення інженерних мереж стає ключовим інструментом вироблення політики модернізації та прийняття рішень. Це забезпечує нормальне функціонування підприємства і досягнення поставлених цілей як у довгостроковій перспективі, так і в питаннях оперативного управління.

Представлений комплекс програмних інструментів, як зазначено у роботі, дозволяє підтримувати актуальні дані про інженерні мережі підприємства та здійснювати моделювання змін в обладнанні. Засоби аналізу та моделювання сприятимуть уникненню помилок, пов'язаних з неузгодженістю роботи та взаємодії обладнання в інженерних мережах. Програмний продукт, запропонований у роботі, не призначений для заміщення проектної документації, але він надає можливість перевірити та змоделювати основні принципи проектних рішень. Вибрана технологія моделювання та принципи редагування моделі можуть застосовуватися для моделювання виробничого процесу з переміщенням сировини та деталей між робочими місцями та обладнанням, де вузлами мережі є пристрої, розподіляють, постачають та використовують воду.

У роботі визначається раціональна стратегія оновлення мереж. Розробка програмного продукту, який дозволить аналізувати результати компонування та з'єднання технічних елементів внутрішніх інженерних

мереж з урахуванням взаємодії цих елементів, є обов'язковим завданням. Для вирішення виникаючих завдань необхідно застосовувати математичні моделі і спеціальні методи для вибору необхідних параметрів.

Мета цього дослідження полягає у розробці програмного засобу, який не лише дозволить експериментально дослідити, але й порівняти принципи формування обґрунтованих основних проектних рішень інженерних мереж. Запропоновані інструменти повинні виконувати важливі завдання пошуку та оновлення інформації. Для досягнення поставленої мети дослідження потрібно вирішити наступні проблеми:

- Дослідити сучасні принципи побудови та сформулювати адекватні задачам моделі контролю та оновлення, необхідні для збереження, відображення та редагування даних.
- Реалізувати ефективні програмні технології для відображення досліджуваних моделей у формі електронних документів.
- Обрати найбільш ефективні та зручні середовища програмування, що вимагає проведення досліджень та порівняльного аналізу сучасних засобів розробки та видів програмної архітектури.
- Обґрунтувати вибір та використати інструментальні та апаратні засоби програмування з локалізацією до української мови.

Завданням запропонованого дипломного проекту є створення програмного засобу, конкретно експертної системи, яка буде забезпечувати актуальну інформацію про інженерні мережі та дозволяти приймати обґрунтовані рішення з питань їх оновлення та модернізації. Основною частиною роботи є розробка інформаційних, програмних та математичних моделей, які слугують основою для прийняття рішень. Важливим аспектом є також розробка інтерфейсу, що максимально відповідає потребам користувачів системи.

Додатково, програмний засіб повинен відповідати стандартним вимогам, серед яких:

- Мінімізація витрат та простота розгортання та впровадження.

- Наглядність та зрозумілість представлення результатів.

Важливим аспектом є забезпечення зручного та ефективного використання системи, а також забезпечення доступності актуальної інформації для прийняття обґрунтованих рішень з питань управління інженерними мережами.

Об'єкт дослідження включає задачі та моделі календарного планування виробництва, принципи проектування мереж, технології контролю використання обладнання та прийняття рішень щодо модернізації. Також розглядаються методи і технології розробки програмного забезпечення та баз даних, а також сучасні та ефективні методи згортки інформації.

Предмет дослідження - гідравлічна мережа або мережа, що визначає послідовність обладнання, необхідного для виготовлення готової продукції, описана за допомогою теорії графів. Дуги визначають можливості транспортування, вершини визначають можливості постачання та перетворення. Задачі аналізу формулюються в термінах відомих моделей дослідження операцій, що базуються на описі мережі. Розробка адекватних алгоритмів для побудови цих моделей є предметом дослідження. Методи аналізу і дослідження реалізовані для конкретного випадку - гідравлічної мережі.

Предметом дослідження також є моделі сформульованих задач та засоби визначення параметрів цих моделей, що будуються відповідно до визначеної схеми.

Практична цінність даного дослідження виявляється у здатності поліпшити якість виконання завдань з обліку та комплектації програмних засобів та матеріальних ресурсів. Програмний продукт, який описано в даній роботі, дозволяє створити модель, придатну для різноманітних задач планування виробництва, і отримати обґрунтований набір параметрів для інженерних мереж.

Елементи наукової новизни виявляються в тому, що описаний програмний продукт може бути застосований для різноманітних задач

планування виробництва, що дозволяє отримати обґрунтований набір параметрів для інженерних мереж.

Практична значущість даної роботи визначається тим, що запропоновані задачі для дослідження є важливою частиною діяльності будь-якого підприємства, спрямованою на оптимізацію та ефективне управління виробничими процесами.

Результати роботи програми були апробовані на прикладах задач проектування та аналізу, що підтверджує їхню відповідність та застосовність у реальних виробничих умовах.

РОЗДІЛ 1 ПОСТАНОВКА ЗАДАЧІ І МЕТОДИ ДОСЛІДЖЕННЯ

1.1. Основні відомості про гідравлічні мережі

Система водопостачання представляє собою комплекс інженерних споруд, призначених для забору води від джерел, її очищення (якщо потрібно), зберігання та подачі до місць споживання. Протипожежна система водопостачання спеціально призначена для постачання води з розрахунковою кількістю та необхідним тиском для ефективного гасіння пожежі. Ця система повинна відповідати встановленим вимогам до надійності та ефективності функціонування всього комплексу системи водопостачання в умовах пожежного ризику.

Вимоги до систем водопостачання визначаються у будівельних нормах, таких як СНиП 2.04.01–85* "Внутрішній водопровід та каналізація будівель" і СНиП 2.04.02 – 84* "Водопостачання. Зовнішні мережі та споруди".

Системи водопостачання можна класифікувати за кількома ознаками. За призначенням поділяють на наступні типи:

- Господарсько-питні системи: Призначені для забезпечення господарських, питних потреб населення та працівників промислових підприємств водою;
- Виробничі системи: Спрямовані на постачання води для потреб виробництва;
- Протипожежні системи: Забезпечують водопостачання для гасіння пожеж;
- Об'єднані системи водопостачання: Можуть включати різні функціональні призначення, такі як господарські, промислові та протипожежні, які об'єднуються в єдину систему.

Ця класифікація дозволяє враховувати різноманітні потреби та функції, які можуть виконувати системи водопостачання в залежності від їхнього призначення.

У містах та населених пунктах часто встановлюють комбіновані системи водопостачання, що об'єднують господарсько-питні водопроводи,

які також забезпечують воду для промислових підприємств. У випадках значних витрат води у промислових підприємств є власні автономні системи водопостачання, які отримують воду як з зовнішніх джерел (з міського магістрального водопроводу), так і з місцевих джерел - поверхневих та підземних. Такі системи водопостачання вирішують завдання забезпечення водою для господарсько-питних, виробничих та протипожежних потреб.

Об'єднання протипожежного водопроводу з господарським має перевагу перед виробничим, оскільки виробнича водопровідна мережа не обов'язково прокладена на кожному об'єкті підприємства. Враховуючи це, деякі технологічні процеси вимагають постачання води під певним тиском, який може змінюватися під час пожежогасіння, що може призвести до аварійних ситуацій.

Отже, для забезпечення пожежної безпеки, гідранти зазвичай встановлюються на господарсько-протипожежних водопроводах. У випадках, коли витрати води на потреби пожежогасіння суттєво менше, ніж для господарсько-питних потреб, гідранти можуть бути встановлені і на господарських водопроводах за потреби. На об'єктах, які вважаються особливо пожежонебезпечними, таких як підприємства нафтохімічної та нафтопереробної промисловості, склади нафти та нафтопродуктів, лісобіржі, сховища зріджених газів і інші, можуть бути влаштовані окремі системи протипожежного водопостачання.

За кількістю об'єктів обслуговування водопроводи класифікуються на наступні типи:

- Централізовані водопроводи: Постачають воду до широкого діапазону об'єктів обслуговування з центральних джерел або магістральних водопровідних систем. Такі системи загальної підтримки використовуються в містах та населених пунктах.
- Місцеві водопроводи: Спрямовані на обслуговування конкретного об'єкта або обмеженої території. Зазвичай використовуються для окремих будинків, підприємств або групи суміжних об'єктів.

- Групові водопроводи: Постачають воду до об'єктів, які об'єднані в групу за територією або функціонально. Це може бути, наприклад, обслуговування кількох сусідніх будинків або підприємств.
- Зонні водопроводи: Постачають воду до конкретних зон або районів. Ця система може бути використана для регіонального обслуговування, де окремі зони мають свої власні водопровідні системи.

Ця класифікація дозволяє враховувати різні рівні масштабу та потреб об'єктів у водопостачанні.

В основному, системи водопостачання будуються у формі централізованих. Ці системи використовуються для надання водопостачання населеним пунктам та промисловим підприємствам. Ці системи є централізованою мережею з одним чи декількома джерелами водопостачання, які забезпечують подачу води до єдиної системи.

Місцеві системи обслуговують окремі будівлі або невелику групу компактно розміщених будівель, отримуючи воду з одного наближеного джерела, такого як промислове підприємство або окремий район міста. Якщо система водопроводу забезпечує водопостачання для кількох об'єктів, таких як групи невеликих населених пунктів або кластер промислових підприємств, її можна назвати груповим водопроводом. Системи водопостачання, які забезпечують різні райони населених пунктів водою під необхідним тиском, особливо враховуючи значну різницю у геодезичних відмітках, можна назвати зонними системами водопостачання.

Централізовані системи водопостачання розподіляються за рівнем надання послуг з подачі води і можуть бути класифіковані в три основні категорії:

- І категорія: дозволяється зменшення подачі води на господарсько-питні та виробничі потреби не більше 30% розрахункової витрати. Період зменшеної подачі не повинен перевищувати 3 дні. Перерви або зменшення подачі можливі під час ремонту та підключення резервних елементів, але не тривають більше 10 хвилин.

- II категорія: передбачає можливість зниження подачі води на господарсько-питні та виробничі потреби не більше 30% розрахункової витрати протягом не більше 10 днів. Перерви в подачі або зниження подачі можуть відбуватися під час ремонту та підключення резервних елементів, проте тривалість цих перерв не повинна перевищувати 6 годин.
- III категорія: передбачає можливість зниження подачі води, як і в I категорії, протягом не більше 15 днів. Тривалість перерв або зниження подачі води може становити до 24 годин у випадку ремонту чи обслуговування.

Об'єднані господарсько-питні та виробничі водопроводи населених пунктів класифікуються за кількістю мешканців:

- I категорія: більше 50 тис. осіб.
- II категорія: від 5 до 50 тис. осіб.
- III категорія: менше 5 тис. осіб.

В системі водопроводу виділяють дві категорії за рівнем тиску:

1. Низького тиску:

- У мережах низького тиску вільний напір на рівні поверхні землі (наприклад, у пожежних гідрантах) повинен становити не менше 10 метрів.
- Для створення необхідного тиску для гасіння пожежі використовують пересувні пожежні автонасоси.

2. Високого тиску:

- У системі високого тиску вода може подаватися безпосередньо від гідрантів.
- Для забезпечення необхідного тиску для гасіння пожежі в мережі та біля стволів використовуються стаціонарні пожежні насоси, розташовані в насосних станціях.

Залежно від типу джерела води, системи водопостачання можуть бути розділені на:

- З забором води з поверхневих джерел: системи, які забирають воду з відкритих водойм, таких як річки, озера або ставки.

- З забором води з підземних джерел: використання води, яка знаходиться в ґрунті або артезіанських свердловинах.
- Зі змішаними джерелами водопостачання: системи, які використовують як поверхневі, так і підземні джерела води.

За методом постачання води, водопроводи можуть бути:

- Напірні (із подачею води насосами): системи, де вода надається за допомогою насосів для забезпечення потрібного тиску.
- Самотічні: системи, де вода рухається гравітаційно, використовуючи різницю в висоті між джерелом і споживачем.

1.2. Постановка задачі

Гідравлічна мережа, або послідовність з'єднаного обладнання для створення готової продукції, може бути аналізована та описана за допомогою теорії графів. Дуги визначають можливості транспортування, а вершини представляють можливості постачання та перетворення. Задачі аналізу формулюються в рамках відомих моделей дослідження операцій, базуючись на даних, що описують структуру мережі. Для створення задачі необхідна інформація, яка подається графічним та інформаційними моделями мережі, та модель формується за допомогою методів теорії графів.

Мета - максимізація надійності та економічності у багатокритеріальній задачі дослідження операцій за критеріями W_1, W_2, ..., W_k. З урахуванням простоти припускаємо, що всі ці параметри слід максимізувати.

Проводиться додатковий аналіз. Розглянемо два можливих рішення, x_1 і x_2, де всі критерії для першого рішення більше або дорівнюють відповідним критеріям для другого, причому хоча б один з них дійсно більший. Очевидно, що у складі множини X немає сенсу зберігати рішення x_2, оскільки його можна витіснити рішенням x_1. Таким чином, викидаємо рішення x_2 як не конкурентоздатне і переходимо до порівняння інших за всіма критеріями. Ця процедура свідомо відкидає непридатні рішення, зменшуючи множину X до так званих ефективних (або "паретовських") рішень, для яких не існує домінуючого варіанту.

У даному випадку виділення "паретовських" рішень стосується задачі з двома критеріями: W_1 і W_2, які обидва мають максимізуватися. Множина X, що представляє собою безліч можливих рішень (x_1, x_2, ..., x_n), зображується на площині з координатами W_1 і W_2. Кожне рішення має визначені значення показників W_1 і W_2, і йому відповідає точка на площині, пронумерована відповідно до номеру рішення.

Для будь-якого іншого рішення існує хоча б одне домінуюче, для якого або W_1, або W_2, або обидва критерії більше, ніж для даного рішення. Лише рішення, що лежать на правій верхній границі, не мають домінуючих альтернатив. Після виділення ефективних рішень "переговори" можуть вестися в межах цього "ефективного" списку.

На прикладі рисунку 1.2.1 чотири рішення (x_2, x_5, x_{10} і x_{11}) формують множину ефективних рішень, де x_{11} вважається найкращим за критерієм W_1, а x_2 - за критерієм W_2. Вибір оптимального варіанту залишається за особою, що приймає рішення, яка обирає той варіант, який відповідає її уподобанням і вважається "прийнятним" за обома критеріями.

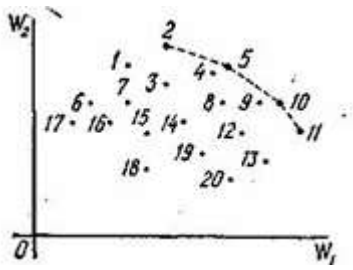


Рис. 1.2.1

Також аналогічна процедура будується для безлічі ефективних рішень, особливо при наявності більш як двох показників (геометрична інтерпретація втрачає наочність при числі показників, більшому за три, але сутність залишається незмінною). Множина ефективних рішень стає більш доступною для аналізу порівняно з множиною X.

Щодо остаточного вибору рішення, він лишається у сфері вирішення особи, що приймає рішення. Лише кваліфікований фахівець, здатний вирішувати неформальні завдання та приймати "компромісні рішення" (не

обов'язково строго оптимальні, але прийнятні за кількома критеріями), може взяти на себе відповідальність за остаточний вибір.

Однак сам процес вибору рішення може виступати основою для розроблення певних формальних правил, які можуть бути застосовані без активної участі людини. Ці правила відомі як "евристичні" методи вибору рішень. Наприклад, якщо досвідчена особа (або кілька досвідчених осіб) систематично обирає компромісне рішення, результати цих виборів можуть бути використані для розробки правил, які визначають ваги різних критеріїв.

Створюючи статистику на основі результатів вибору, можна розумно визначити значення "ваги" (a_1, a_2, ...) для кожного критерію (W_1, W_2, ...) залежно від умов і самих показників. Ці ваги можна використовувати для автоматизованого вибору рішення без прямого втручання людини. Цей процес може бути особливо корисним в ситуаціях, коли немає часу для обдумування компромісного рішення або коли вибір рішення передається автоматизованій системі.

У випадку нашого дослідження, ефективний спосіб вибору рішення може виконуватися через "діалоговий режим". Після проведення розрахунків машина передає значення показників W_1, W_2, ..., особі або групі осіб, які керують операцією. Критично оцінюючи ситуацію, ці особи можуть вносити зміни у вагові коефіцієнти або інші параметри керуючого алгоритму.

Існує інший підхід, який зводить багатокритеріальну задачу до задачі з одним критерієм. Цей метод полягає в тому, щоб виділити один основний показник W_1 та спрямовувати його на максимум, при цьому залишаючи лише обмеження для інших показників W_2, W_3, ... щодо їхніх мінімальних значень (наприклад, w_2, w_3, ...). Наприклад, при оптимізації плану роботи мережі можна вимагати, щоб прибуток був максимальним, план постачання був виконаний або перевиконаний, а собівартість постачання не перевищувала заданого рівня. Цей підхід дозволяє перетворити багатокритеріальну задачу на оптимізацію за одним головним критерієм, з обмеженнями на інші.

Ці обмеження, такі як w_2, w_3, ..., можуть бути введені в "діалоговому режимі" за участю осіб, які керують операцією. Це надає можливість активного втручання та коригування параметрів з урахуванням конкретних обставин чи вимог.

Існує інший метод побудови компромісного рішення, який можна назвати "методом поступок". Цей підхід передбачає розташування показників W_1, W_2, ... в порядку зменшення їхньої важливості. Починаючи з пошуку рішення, яке максимізує перший (найважливіший) показник $W_1=W_1*$, здійснюється "поступка" W_1, враховуючи практичні обставини та можливі помилки вхідних даних. Ця "поступка" дозволяє максимізувати другий показник W_2.

За умови, що W_1 не менше, ніж W_1* - W_1, відповідне обмеження трансформується у максимум для W2. Після цього призначається "поступка" в W_2, яка, в свою чергу, дозволяє максимізувати W_3, і так далі. Цей процес триває, забезпечуючи поетапне максимізування кожного показника.

Такий метод конструктивний, оскільки він дозволяє чітко бачити, який виграш в одному показнику може бути отриманий за рахунок "поступки" в іншому. Це дозволяє приймати інформовані рішення з урахуванням конкретних обставин та важливості кожного показника.

У будь-якому випадку, незважаючи на обрані методи, завдання обґрунтування рішення проекту мережі за кількома показниками залишається не до кінця формалізованим, і остаточний вибір рішення завжди визначається вольовим актом проектувальника (так можна умовно назвати відповідальну за вибір особу). Мета запропонованої системи - надати проектувальнику дані, що допомагають йому приймати обґрунтовані рішення, ураховуючи переваги і недоліки кожного варіанту рішення.

Завдання проектування мережі поділяються на дві категорії: прямі і зворотні. Прямі завдання відповідають на питання: що буде, якщо в заданих умовах ми приймемо певне конструктивне рішення мережі, позначене як x. Зокрема, вони визначають, яке значення матиме обраний показник

ефективності W (або ряд таких показників) при прийнятому рішенні x.

У нашому випадку, типова задача першої категорії для гідравлічних мереж полягає в розрахунку потоків. Для вирішення цього завдання створюється математична модель, яка дозволяє виразити один або кілька показників ефективності через задані умови та елементи рішення. Результат розрахунку полягає у визначенні потоку на кожній ділянці мережі, що відповідає умовам нерозривності потоку:

$$\sum_{i \subset W_j}(g_j) = 0, W_j \subset MW \quad (1)$$

та втрат напору для кожного маршруту на мережі, заданим:

$$\sum_{i \subset M_j} f(g_j) = H_j, M_j \subset MM \quad (2)$$

де MW_j та MM_j. відповідають множинам маршрутів та вузлів мережі відповідно.

Для прямої задачі, система рівнянь є еквівалентною задачі оптимізації:

$$\left(\sum(g_j)\right)^2 + \left(\sum f(g_j) - H_j\right)^2 \to min \quad (3)$$

за умови $g_j \geq 0$. У багатьох випадках необхідні додаткові умови:

$g_j \geq G_i$.

де G_i - певні порогові значення.

Забезпечення ефективності системи завжди передбачає потребу в мінімізації витрат на її модернізацію. Зворотні задачі виникають, коли метою є вибір оптимального рішення (позначеного як x), яке максимізує показник ефективності W.

Зазвичай прямі задачі є більш простими, і вирішення оберненої задачі передбачає спочатку вирішення прямої. Деякі операції вирішуються настільки легко, що їх навіть не розглядають окремо. Але для інших операцій побудова математичних моделей та обчислення показників ефективності є складним завданням, і остаточний вибір рішення завжди є результатом особистого рішення проектувальника.

Якщо маємо невелике число можливих варіантів рішення, що формують множину X, можемо використовувати метод "простого перебору",

обчислюючи значення показника W для кожного варіанту і порівнюючи їх для визначення оптимальних рішень, де W максимальне. Однак у випадках, коли можливих варіантів рішення є велика кількість або навіть безліч, застосування простого перебору стає непрактичним або неможливим. У таких ситуаціях використовують методи "спрямованого перебору", які дозволяють ефективніше визначити оптимальні рішення, враховуючи певні напрямки або наближення.

Отже, ми розглядаємо операцію проектування мережі, позначену як O, де успіх операції визначається показником ефективності W, який ми хочемо максимізувати. Вводяться дві групи факторів:

1. Задані фактори (означені як "a"): Це умови виконання операції, які заздалегідь відомі і не містять невизначеностей. Ця група включає обмеження, які накладаються на рішення і визначають область можливих рішень, позначену як X.

2. Фактори, що залежать від проектувальника (позначені як "x"): Це параметри, які можна вибирати для впливу на успіх операції. Рішення проектувальника формується як група параметрів "x".

У "детермінованому" випадку, коли умови операції повністю відомі заздалегідь і не містять невизначеностей, ми спрямовані на максимізацію показника W.

Таким чином, показник ефективності W залежить від обох груп факторів, і це можна виразити у вигляді формули:

W = W(a, x)

При цьому важливо зауважити, що як x, так і *a* в загальному випадку представляють не просто числа, а сукупність проектних елементів, функцій і т. д. Обмеження для заданих умов *a* часто виражаються у вигляді рівностей або нерівностей, які накладаються на елементи рішення.

Задача вже вирішена у прямому вигляді згідно з рівняннями (1) – (3). Тепер зворотне завдання формулюється наступним чином:

при фіксованому комплексі умов *a* знайти рішення x = x*, яке

максимізує показник ефективності W і досягає максимального значення W*. Це W* є максимальним значенням W(a, x) серед усіх можливих рішень X.

Отже, у реальних задачах проектування мереж часто присутня ще одна група факторів - невідомі фактори, які можна позначити символом ξ. Отже, показник ефективності W залежить від трьох груп факторів:

$$W = W(\alpha,\ x,\ \xi).$$

Оскільки W залежить від невідомих факторів ξ, то навіть при фіксованих значеннях *a* і x вона залишається невизначеною. Задача пошуку оптимального рішення стає більш складною через випадковий характер цих факторів. Ми можемо взяти середнє значення (математичне очікування) цієї випадкової величини, позначивши його як $\hat{W} = M[W]$ і шукати рішення x, при якому це середнє значення показника ефективності максимізується:

$$\overline{W} = M\,[W\,(\alpha, x, \xi)] \Rightarrow \max.$$

При розгляді задач, що містять невизначеність, показник ефективності не просто оцінюється як "дохід" чи "час", а як "середній дохід" чи "середній час". Врахування середнього дозволяє більш точно враховувати випадковий характер факторів.

В більшості випадків оптимізація в середньому є виправданою, оскільки вибір рішення, яке максимізує середнє значення показника ефективності, вважається більш обґрунтованим, ніж вибір рішення наугад. Невизначеність залишається, оскільки ефективність кожної окремої операції при конкретних значеннях випадкових факторів може суттєво відрізнятися від очікуваної.

У випадку обмежень на область зміни незалежних змінних, важливо визначити, які умови можна накладати на ці змінні. Наприклад, можна вимагати, щоб умова $G_i \le t_0$ виконувалася з великою ймовірністю, зробивши подію $G_i \le t_0$ практично достовірною.

Методи дослідження задач типу (1) – (3) при наявності обмежень на область зміни незалежних змінних можуть бути використані для пошуку

окремих екстремальних значень всередині області зміни, зокрема в задачах дослідження мереж. Особливо це стосується задач з великою кількістю незалежних змінних, де аналіз значень критерію оптимальності на межах допустимої області може бути складним.

1.3. Опис середовища розробки

Середовище візуальної розробки програм Microsoft Visual Studio, також відома як IDE (Інтегроване середовище розробки), призначене для створення, відкриття, перегляду, редагування, збереження, компіляції, побудови та відладки додатків. Воно включає в себе інструменти для розробки мовами програмування, такими як Visual Basic, Visual C++, Visual C#, Visual F#.

Після запуску Microsoft Visual Studio з'являється головне вікно IDE, яке включає стартову сторінку (рис. 1.3.1).

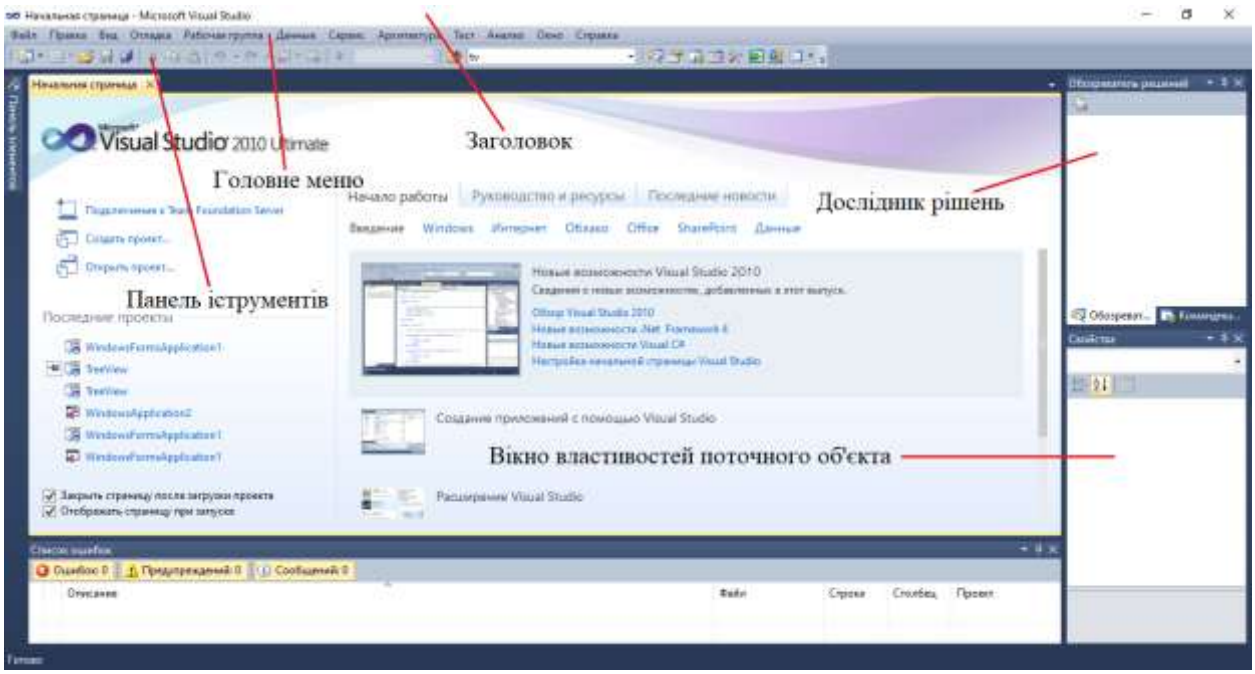


Рис. 1.3.1. Стартова сторінка

Активація команд меню в Microsoft Visual Studio може бути виконана різними способами:

- Гарячі клавіші: Використовуйте гарячі клавіші, які вказані поруч із відповідними пунктами меню. Для цього натискайте клавішу Alt разом із підкресленою буквою у пункті меню.

- Миша: Активуйте меню, натиснувши на нього мишею, і виберіть пункт меню, клацнувши на ньому.
- Панель інструментів: Використовуйте відповідну кнопку (піктограму) на панелі інструментів, яка розташована ліворуч від відповідної команди.

Активація меню також може бути виконана за допомогою клавіш швидкого виклику, таких як Alt разом із підкресленою буквою в пункті меню.

У випадкових меню можуть бути команди, які виділені блідо-сірим кольором, що означає їхню недоступність у даному контексті через відсутність необхідних умов. Якщо назва команди має три крапки, це свідчить про те, що при її виборі відкриється діалогове вікно. Якщо є темний трикутник, це може вказувати на наявність підменю.

Головне меню включає такі пункти:

1. Меню "File" включає в себе команди, пов'язані з управлінням файлами. Ці команди дозволяють створювати нові файли та проекти, відкривати існуючі, закривати, зберігати та друкувати їх. Крім того, ви можете додавати нові чи існуючі проекти та елементи (такі як форми, класи, файли тощо) до вашого рішення.

У нижній частині меню розташовані назви нещодавно відкритих файлів та проектів. Ця функція полегшує швидкий доступ до будь-якого з них. Останньою командою у меню "File" є "Exit" (Вихід), призначена для виходу з Visual Studio. Вона забезпечує обробку будь-яких несхованих змін перед закриттям програми.

2. Меню "Edit" є стандартним у багатьох додатках Windows та використовується для редагування текстового та програмного коду. У цьому меню розміщені стандартні команди редагування, такі як скасувати попередню дію, повернутися до попередньої дії, вставити, вирізати фрагмент тексту, виділити весь текст, видалити виділене, а також команди пошуку і заміни фрагменту тексту та переміщення до певного рядка.

За допомогою команд пункту IntelliSense у меню "Edit" можна переглядати список членів конкретного класу, структури, об'єднання або простору імен та вставляти в код відповідний фрагмент. Меню також дозволяє отримувати інформацію про кількість, типи і імена параметрів методів і властивостей, а також автоматично доповнювати імена методів та команд.

Команда "Insert Snippet" у меню "Edit" IntelliSense відображає ієрархічний список найбільш часто виконуваних завдань. Обравши один із пунктів списку, в програму вставляється фрагмент коду, що спрощує виконання конкретного завдання.

Це меню також може містити команди, які дозволяють виділяти текст як коментар, показувати недруковані символи, встановлювати перенесення слів, робити всі виділені символи малими або великими та інші.

3. Меню "View" містить команди, що стосуються відображення та організації робочого середовища Visual Studio. За допомогою цих команд можна відкривати різні вікна, зокрема редактор програмного коду, конструктор форм, вікна властивостей об'єктів, оглядача рішень та інші.

Команда "Toolbars" надає можливість користувачеві змінювати панелі інструментів, щоб пристосувати інтерфейс до власних потреб. Вона дозволяє активувати або вимикати різні панелі інструментів, залежно від того, які функції важливі для користувача у даному контексті.

У нижній частині меню "View" розташована команда "Property Pages", яка відкриває вікно для визначення загальних налаштувань поточного проекту. Це вікно дозволяє встановлювати різноманітні параметри та властивості проекту, щоб краще відповідати вимогам розробки.

4. Меню "Project" включає команди, спрямовані на управління елементами проекту у Visual Studio. Ці команди дозволяють додавати та видаляти різні елементи, такі як форми, програмні модулі, класи, і також забезпечують можливість додавати посилання на спільні бібліотеки.

Команда "Properties" є останньою у меню "Project" і відкриває вікно властивостей проекту. У цьому вікні можна настроювати різні параметри та властивості проекту, такі як налаштування компіляції, властивості виконання, асоціації файлів та інші параметри, які впливають на процес розробки та виконання проекту.

5. Меню "Build" у Visual Studio містить команди, які надають доступ до інструментів, спрямованих на генерацію коду, відладку та запуск створеної програми. Однією з ключових команд у цьому меню є "Rebuild Solution".

Команда "Rebuild Solution" є однією з найважливіших у меню "Build". Ця команда з'являється після створення проекту. Вона використовується для повної перекомпіляції та перебудови всього рішення (solution).

Загальна функціональність команд у меню "Build" спрямована на оптимізацію процесу компіляції та забезпечення правильної роботи створеної програми.

6. Меню "Debug" у Visual Studio містить команди, спрямовані на налагодження та запуск додатка. Це меню дозволяє виконувати різні операції для контролю за виконанням програми під час налагодження.

7. Меню "Format" у конструкторі форм Visual Studio містить команди, які спрощують вирівнювання та оформлення об'єктів на формі. Це особливо корисно при роботі з інтерфейсом користувача. Внизу цього меню розташовується команда "Lock Controls", яка фіксує розташування та розмір обраних елементів управління, включаючи саму форму. Це перешкоджає випадковим змінам їхнього положення чи розміру.

Меню "Format" спрощує процес розміщення та оформлення об'єктів на формі, забезпечуючи зручні інструменти для створення естетичного та функціонального інтерфейсу користувача.

8. Меню "Tools" у Visual Studio включає в себе інструменти, які допомагають налаштовувати середовище розробки, створювати макроси та використовувати додаткові утиліти. Воно є ключовим елементом для

налаштування та розширення функціональності Visual Studio, роблячи розробку більш ефективною та зручною.

9. Меню "Window" у Visual Studio містить команди для керування відкритими вікнами на екрані. Це дозволяє вам організовувати та навігувати серед вікон для зручної роботи. Це меню надає зручний спосіб керування великою кількістю відкритих вікон і сприяє більш ефективній організації робочого простору.

10. Меню "Help" у Visual Studio містить команди та інструменти для отримання довідкової інформації та підтримки. Воно допомагає користувачам отримати докладну інформацію, знайти відповіді на питання, а також взаємодіяти зі спільнотою та звітувати про проблеми для подальшого вдосконалення розробленого програмного забезпечення.

1.4. Висновок до розділу

У даному розділі ми оглянули концепцію гідравлічної мережі та її призначення. Також, ми детально розглянули основні аспекти середовища розробки MS Visual Studio. Ми представили модель, яка враховує умови задачі, та вказали, що параметри цієї моделі можуть бути знайдені шляхом розв'язання задачі контролю або за допомогою методу експертних оцінок.

Наш підхід включав розробку моделі задачі призначення, використання алгоритму на основі методу гілок та меж. Цей метод є актуальним для розв'язання складних задач і може знайти застосування в широкому спектрі ситуацій. Ми також представили нову модель задачі, враховуючи особливості евклідової комбінаторної оптимізації.

Висновок полягає в тому, що наш підхід до моделювання та розв'язання задачі щодо призначення є перспективним і враховує необхідні особливості самої задачі.

РОЗДІЛ 2 ОБҐРУНТУВАННЯ МОДЕЛЕЙ ТА МЕТОДІВ

Сформульована модель (моделі) задачі вибору проектних параметрів мережі належить до найскладніших задач математичного програмування, які відомі як NP-повні. У нашому випадку виникла NP-повна задача, для вирішення якої доводиться проводити аналіз та знаходити задовільний розв'язок. В таких випадках важливим стає діалоговий режим роботи, а іноді можливість скорочення повного перебору так, що етапи алгоритму, залишаючись в гіршому випадку експоненціальними, можуть працювати за прийнятний час на реальних даних. Основою технології, що пропонується, є методи перетворення моделей, які відповідають задачам.

Перша модель, яку будує система, є графічною, наочною і зрозумілою (рис. 2.1).

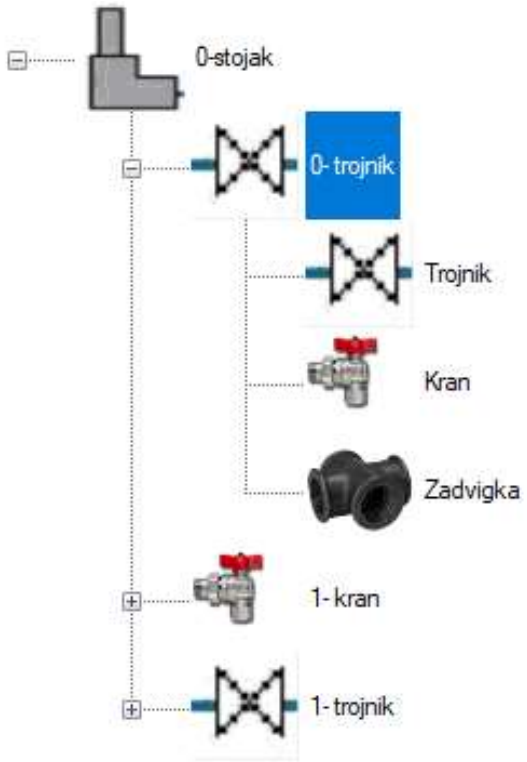


Рис. 2.1

Друга реляційна модель даних, що базується на рекурсивних зв’язках (рис. 2.2), використовується в якості основи для алгоритмів перетворення та побудови моделей. Ці алгоритми базуються на методах теорії графів, яка є математичним поняттям для моделювання зв'язків між об'єктами.

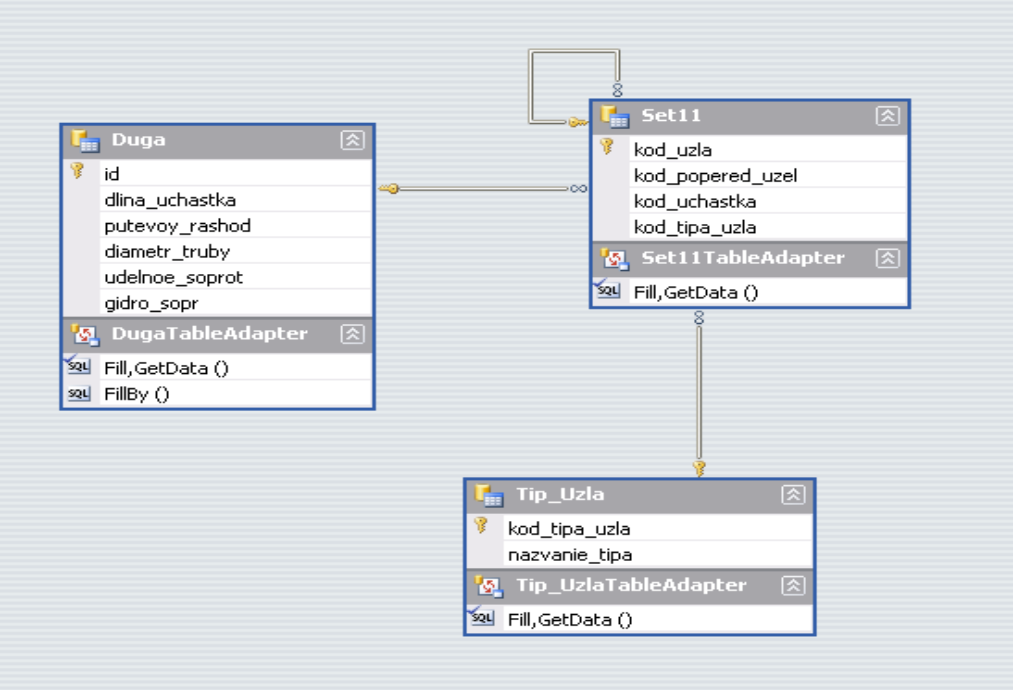


Рис. 2.2

2.1 Графи

Графом G називається пара множин G = (X, W), де X – множина вершин, і W – множина ребер(рис. 2.1.1).

Мультиграф – граф, в якому одну й ту саму пару вершин з'єднує кілька ребер(рис. 2.1.2).

Граф з петлями – граф, в якому є ребра, що мають збіжні кінці(рис. 2.1.3).

Псевдограф – мультиграф з петлями(рис. 2. 1.4).

Орієнтований граф – граф, ребра якого мають напрямок(рис. 2.1.5).

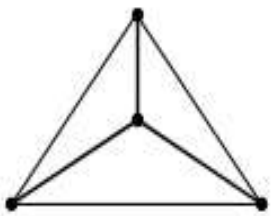


Рис. 2.1.1

Рис. 2.1.2

Рис. 2.1.3

Рис. 2.1.4

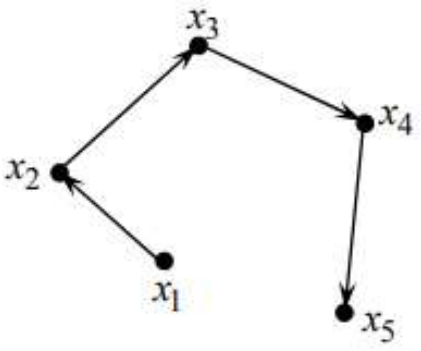


Рис. 2.1.5

Розмір графа G визначається кількістю його вершин, що позначається як n, та його потужністю, що позначається як m. Ребро w = (x, y) є з'єднанням вершин x та y як його кінців. Вершини x і y є суміжними, і кожна з них інцидентна ребру w. Різні ребра, що інцидентні одній і тій же вершині, називаються суміжними.

Граф завдання будується за такою схемою: вершини представляють споживачі, джерела та пристрої для розгалуження; ребра відповідають трубам; вага ребра визначається довжиною ділянки та технологією обрахунку гідравлічного опору; вага вершини представляє тиск.

Задамо натуральні номери від 1 до n для всіх вершин графа G. Матриця суміжності A графа G - це квадратна матриця розміром n×n. У цій матриці a_{ij} що відповідає елементу в i-му рядку та j-му стовпчику, рівне 1, якщо вершини x_i та x_j з номерами i та j є суміжними, і рівне 0, якщо вони не суміжні.

$$a_{ij} = \begin{cases} 1, & (x_i, x_j) \in W, \\ 0, & (x_i, x_j) \notin W; \end{cases}$$

Маршрут в графі G = (X, W) - це послідовність ребер, де кожні два сусідні ребра, позначені як w_{i-1} та w_i мають спільну вершину x.

Зв'язний граф - це граф, у якому існує маршрут, що з'єднує будь-які дві вершини.

Маршрут є замкненим, якщо перше ребро (w_0) дорівнює останньому ребру (w_n).

Ланцюг - це маршрут, в якому всі ребра є різними.

Простий ланцюг - це ланцюг, в якому всі вершини є різними.

Цикл - це замкнений ланцюг.

Простий цикл - це цикл, в якому всі вершини, крім першої та останньої, є різними.

Ейлеровий цикл - це цикл, у якому кожне ребро графа використовується один раз. Граф, що містить такий цикл, називається ейлеровим графом.

Леонард Ейлер був першим, хто поставив і відповів на питання про те, чи можна пройти кожен міст у місті один раз і повернутися в початкову точку (див. рис. 2.1.6). Це формулюється як задача пошуку ейлеревого циклу в графі.

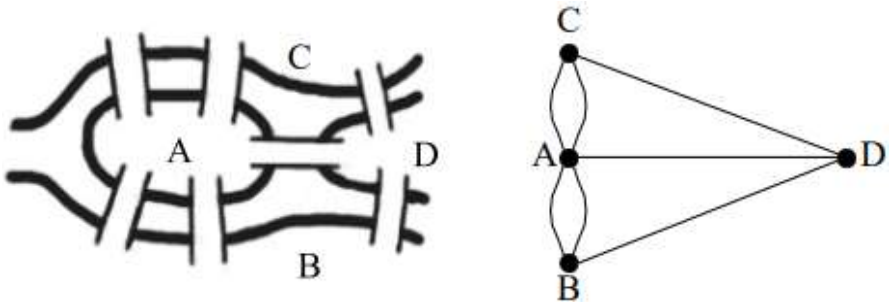


Рис. 2.1.6. кенігсберзькі мости та граф

Теорема Ейлера стверджує: "Скінчений неорієнтований граф G є ейлеровим тоді і тільки тоді, коли він зв'язний і ступені всіх його вершин парні."

Алгоритм побудови ейлерового циклу виглядає наступним чином:

1. Перевірка графа відповідно до теореми Ейлера. Якщо граф не відповідає умовам, то ейлерового циклу не існує.

2. Вибір початкової вершини x_0.

3. Вибір довільного ребра w, що інцидентне x_0, і надання йому номера 1 (позначеного як пройдене ребро).

4. Видалення кожного пройденого ребра та присвоєння йому номера, на одиницю більшого за номер попереднього видаленого ребра.

5. При перебуванні в вершині x_i, уникання вибору ребра, що з'єднує вершини x_i та x_0, якщо є інший варіант.

6. При перебуванні в вершині x_i, уникання вибору ребра, яке є перешийком (тобто ребро, видалення якого призводить до розпаду графа на дві компоненти зв'язності).

7. Завершення алгоритму після пронумерованих всіх ребер, утворюючи Ейлерів цикл, і порядок нумерації відповідає послідовності обходу ребер.

2.2 Дерева

Дерево - це зв'язний граф без циклів. У дереві відсутні кратні ребра, петлі та ізольовані вершини. Кожен ланцюг у дереві є простим, оскільки в іншому випадку граф містив би цикл, що є неможливим (див. рис. 2.2.1).

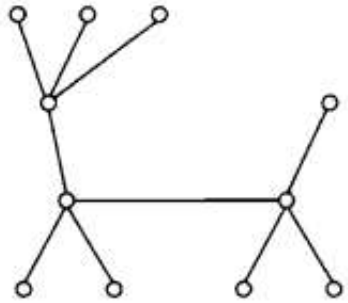


Рис. 2.2.1

Якщо граф G(X, W) має n вершин і m ребер, то наступні умови є еквівалентними:

1. G(X,W) – дерево;
2. G(X,W) – зв’язний граф і m = n −1;
3. G(X,W) – ациклічний граф і m = n −1;
4. Будь-які дві вершини графа G(X, W) сполучені єдиним простим ланцюгом;
5. G(X,W) – ациклічний граф, який має властивість: сполучивши ребром будь-яку пару його несуміжних вершин, отриманий граф міститиме рівно один цикл.

Граф $G_1 = (X_1, W_1)$ називається підграфом графа $G = (X, W)$, якщо множина вершин X_1 є підмножиною X, а множина ребер W_1 є підмножиною

W. Переносити до графа G_1 означає включати в нього певні вершини та ребра вихідного графа.

Кістяковий підграф графа G = (X, W) визначається як $G_1 = (X_1, W_1)$, де множина вершин X_1 повністю співпадає з множиною вершин G ($X_1 = X$), і множина ребер W_1 є підмножиною множини ребер G ($W_1 \subseteq W$). За умови, що вилученням лише деяких ребер графа G, не зачіпаючи вершин, можна отримати кістяковий підграф.

Вага ребра - це числове значення, яке призначається кожному ребру графа і відображає його певні характеристики.

Вага дерева обчислюється як сума ваг усіх його ребер.

2.3 Алгоритми побудови мінімального кістякового дерева

В ситуаціях проектування доріг, електричних мереж, трубопроводів тощо, коли потрібно пов'язати безліч об'єктів комунікаційними лініями з мінімальною сумарною вартістю, виникає задача пошуку остовного дерева мінімальної ваги в графі. Це означає, що потрібно знайти такий підграф (остовне дерево), який включає всі вершини графа та є деревом з мінімальною можливою сумою ваг ребер.

Алгоритми, які вирішують цю задачу, будують кістяк мінімальної ваги для графа G = (X, W), де X - множина вершин, W - множина ребер.

Алгоритм Крускала

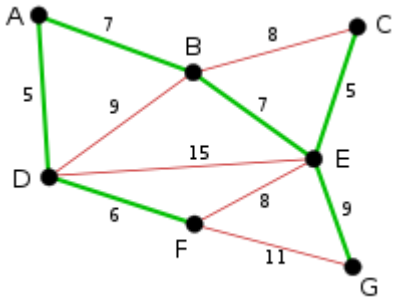
Алгоритм Крускала є методом для створення мінімального кістякового дерева у зваженому неорієнтованому графі. Вперше цей алгоритм було представлено Джозефом Крускалом у 1956 році.

Алгоритм Крускала:

1. Спочатку множині W присвоюється значення порожньої множини, позначене символом ∅.
2. Визначаємо, які ребра початкового графа, що не входять до множини W, можна додати до неї без утворення циклів.
3. Якщо такі ребра існують, то:

- Обираємо серед них "найлегше" ребро, тобто те яке має найменшу вагу.
- Додаємо це ребро до множини W.
- Повторюємо процес, переходячи до виконання другого кроку.

4. Якщо не існує ребер, які можна додати без утворення циклів, завершуємо побудову мінімального остовного дерева.



Послідовність обраних ребер: AD, CE, DF, AB, EB, EG

Вага дерева: 5 + 5 + 6 + 7 + 7 + 9 = 39

Алгоритм Крускала є ефективним способом знаходження мінімального остовного дерева та знаходження оптимального шляху для побудови комунікаційних мереж з мінімальною вартістю.

Алгоритм Прима

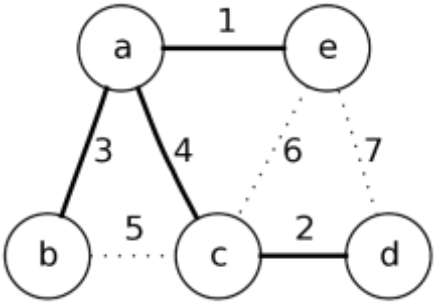
Алгоритм Прима - це жадібний метод створення мінімального кістякового дерева у зваженому зв'язаному неорієнтованому графі.

Алгоритм Прима:

1. Спочатку утворюємо дерево T_1, яке складається з одного ребра:
   - Вибираємо довільну вершину x_0 з початкового графа G.
   - Обираємо ребро w_1 з найменшою вагою серед тих, що з'єднуються з вершиною x_0.
   - Позначаємо k = 1.
2. Якщо існують вершини в початковому графі G, які не належать останньому побудованому дереву T_k з ребрами w_1, w_2, w_3, ..., w_k, то виконуємо наступне:

- Вибираємо ребро w_{k+1} з найменшою вагою серед тих, у яких одна вершина належить до T_k, а інша не належить.
- Формуємо дерево T_{k+1}, додаючи до T_k обране ребро w_{k+1} і його вершини.
- Збільшуємо k на 1
- Повертаємося до початку виконання пункту 2.

3. Якщо всі вершини початкового графа G належать дереву T_k, завершуємо побудову мінімального остовного дерева.



Послідовність обраних ребер: ae, ab, ac, cd

Вага дерева: 1 + 3 + 4 + 2 = 10

Алгоритм Борувки

Алгоритм Борувки - це метод пошуку мінімального кістякового дерева у графі, який був вперше описаний Отакаром Борувкою в 1926 році як засіб пошуку оптимальної електричної мережі в Моравії.

Алгоритм Борувки:

1. Спочатку маємо порожню множину ребер T (яка представляє собою ліс, де кожна вершина є окремим деревом).

2. Доти, поки множина T не стане деревом (тобто, кількість ребер у T менше, ніж кількість вершин X - 1, де X - кількість вершин у графі):

- Для кожної компоненти зв'язності (тобто, дерева в кістяковому лісі) у підграфі з ребрами T, знаходимо найдешевше ребро, яке з'єднує цю компоненту з будь-якою іншою компонентою зв'язності. Вважаємо, що вага ребер різна або впорядкована так, що завжди можна знайти єдине ребро з мінімальною вагою.

- Додаємо знайдені ребра до множини T.

3. Отримана множина ребер T є мінімальним кістяковим деревом початкового графу.

Алгоритм Борувки вважається ефективним для реалізації на практиці, особливо для розподілених систем або паралельних обчислень, оскільки він може легко паралелізуватися.

Алгоритм Дейкстри

Алгоритм Дейкстри - це метод на графах, розроблений Дейкстрою, для пошуку найкоротшого шляху від однієї вершини графу до всіх інших вершин. Класичний алгоритм Дейкстри працює лише для графів без ребер від'ємної довжини.

Розглянемо орієнтований граф G = (X, W) з виділеною вершиною s та вагами ребер w_{ij}, $(i, j) \in X$.

При використанні цього алгоритму застосовуються такі правила:

1) Вершинам присвоюються значення відстаней від виділеної вершини.

2) Остаточні позначки присвоюються вершинам, коли множина шляхів не складається з усіх можливих.

3) Алгоритм завершує роботу, коли позначка кінцевої вершини не змінюється.

4) На кожному кроці деякі вершини мають остаточні позначки, а інші - тимчасові.

Алгоритм Дейкстри виглядає наступним чином:

1. Вибирається початкова вершина графа та присвоюється їй нульова відстань. Тобто, початковій вершині присвоюється остаточна позначка "0", а іншим вершинам - тимчасова позначка "?".

2. Для кожної вершини перебираються всі її сусіди (вершини, з'єднані ребрами), і кожній з них присвоюється відстань, утворена шляхом додавання ваги відповідного ребра до відстані від поточної вершини. Кожній вершині x, яка ще не отримала остаточну позначку, присвоюється нова

тимчасова позначка за правилом: $\min(d_x, d_y+w_{yx})$, де d_x - стара тимчасова позначка вершини x; y - вершина, яка одержала остаточну позначку на попередньому кроці; w_{yx} - вага дуги (y,x).

3. Знаходиться найменша з усіх тимчасових позначок, і ця вершина отримує остаточну позначку. Вона стає вершиною для наступного кроку.

4. Якщо сусідній вершині вже присвоєно якесь значення, то залишається те, яке є меншим. В результаті цієї операції для кожної вершини отримується число, що вказує на мінімальну відстань від початкової вершини до даної.

5. Знаючи мінімальну відстань до кожної вершини, можна послідовно переглядати сусідів і дійти до початкової вершини, знаходячи таким чином шлях.

Алгоритм Дейкстри працює добре для графів з невеликою кількістю вершин, але може бути менш ефективним для великих графів у порівнянні з іншими алгоритмами, такими як алгоритм Беллмана-Форда.

2.4. Висновок до розділу

У цьому розділі ми сформулювали модель нашої задачі вибору проектних параметрів мережі, що відноситься до найскладніших задач математичного програмування, відомих як NP-повні. Була розглянута основна інформація про графи та дерева.

Основою системи є алгоритми перетворення та побудови моделей, які базуються на методах теорії графів. Граф задачі конструюється так, що вершини представляють споживачі, джерела та пристрої для розгалуження, а ребра відображають труби. Вага ребра визначається довжиною ділянки та технологією обчислення гідравлічного опору, в той час як вага вершини представляє тиск.

Чотири алгоритми пошуку графами та деревами, а саме алгоритм Прима, алгоритм Крускала, алгоритм Борувки та алгоритм Дейкстри, були розглянуті та порівняні з точки зору їхніх переваг та недоліків. Після аналізу

було вирішено використовувати алгоритм Дейкстри для вирішення поставленої задачі.

РОЗДІЛ 3 АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Обґрунтування принципів програмної реалізації

Системні алгоритми та методи дослідження задачі втілюються з використанням принципів, що визначаються стандартами шаблонів програмування на платформі технологій .NET та відповідно до концепцій Model-View-Controller (MVC). Проект програмної системи ґрунтується на архітектурній схемі Model-View-Controller. Основна ідея та принцип полягають в чіткому розділенні етапів обробки даних, і проект поділений на три ключові компоненти, кожна з яких відповідає за різні завдання, відповідно до наступної схеми.

Контролер в даній системі керує командами проектувальника, які надходять у вигляді запитів HTTP GET або POST, коли користувач взаємодіє з графічним інтерфейсом для виконання певних дій. Основна функція контролера полягає в виклику та координації функцій об'єктів, необхідних для виконання завдань користувача. Контролер визначає необхідну модель для завдання та визначає правильний вигляд (View).

Модель представляє собою дані та правила, які застосовуються до цих даних, що відображають концепції інженерної мережі, які керує додатоком. У розробленій програмній системі поняття інженерної мережі моделюється як дані, що підлягають обробці. Модель інженерної мережі включає найважливішу частину логіки системи, що стосується проблеми інженерної мережі. Ця модель даних є незалежною від того, яким чином її можна представити графічно, тому доступні різні варіанти відображення даних. Модель включає всі необхідні дані та важливу частину логіки програми, і її структура формується за допомогою засобів CASE-технологій, таких як ERWin.

Основою моделі є рекурсивні зв'язки між таблицями, що дозволяють ефективно моделювати мережі. Це досягається завдяки сполученню таблиць через зв'язки, які можуть бути реалізовані у вигляді різних типів зв'язків, таких як один до одного, один до багатьох та багато до багатьох.

ERWin використовується для створення моделей, аналізу та розробки інформаційних систем. Цей інструмент також служить для автоматичного генерування системного коду для баз даних, серверного коду та клієнтських додатків. Щоб відокремити функції, ERWin використовує два типи моделей даних: логічну та фізичну.

Логічна модель є незалежною від конкретної системи керування базами даних (СКБД) і представляє об'єктну декомпозицію предметної області, яка вивчається для створення інформаційної системи. У свою чергу, фізична модель враховує конкретні особливості обраної СКБД та структуру технічних засобів.

Важливо відзначити, що кожна логічна модель може мати кілька реалізацій у вигляді фізичних моделей, які залежать від обраної СКБД в ERWin.

ERWin надає інструменти для створення наглядної логічної схеми або моделі бази даних, а також допомагає розробнику провести її дослідження, оптимізацію та узгодження з замовником. Система використовує три рівні логічної моделі даних:

- Діаграма об'єкт-зв'язок (ERD) є високорівневою моделлю, спрямованою на обговорення структури даних з експертами та замовником. На цьому рівні об'єкти бази даних зображаються у вигляді прямокутників, а зв'язки між ними визначають структуру даних.
- Модель даних, що базується на ключах (Key Based Model): На цьому рівні модель зосереджується на ключах та відносинах між ними. Визначаються ключі, які визначають, як дані будуть організовані та зберігатися в базі даних.
- Повна атрибутивна модель (Fully Attributed Model): Цей рівень додає деталі та атрибути до ключів, визначаючи конкретні аспекти даних, такі як їхні типи та обмеження.

Кожен з цих рівнів логічної моделі даних служить певною метою в розробці бази даних і дозволяє розробникам працювати на високому та

деталізованому рівнях для визначення та оптимізації структури даних відповідно до потреб проекту.

Діаграма об’єкт-зв’язок є високорівневою моделлю даних, спрямованою на обговорення структури інформації з експертами з боку замовника. Під час створення цієї моделі об'єкти бази даних представлені у вигляді прямокутників, в яких містяться визначення збережених об’єктів. Взаємозв'язки між об'єктами формуються відповідно до бізнес-логіки функціонування організації, і цю схему обговорюють та погоджують з замовником, часто з включенням юриста.

Діаграма об'єкт-зв'язок складається з основних компонентів, а саме об'єктів, або сутностей. Кожен об'єкт відображає самостійний набір інформації з предметної області. Назва об'єкта вказується у формі іменника у однині, що відповідає сутності одного запису в базі даних. Наприклад, об'єкт може мати назву "Замовник" та включати атрибути, такі як прізвище, номер та адреса замовника.

Об'єкт представляє конкретний екземпляр сутності, і сама сутність має те саме ім'я, що і об'єкти, які до неї входять. Наприклад, сутність, яка відповідає множині замовників продукції підприємства, може мати назву "Замовник", так само, як і об'єкт замовник, оскільки вона має ті ж самі атрибути.

Таким чином, це означає, що на рівні об'єктів і сутностей діаграма об'єкт-зв'язок буде виглядати ідентично. З врахуванням реляційної бази даних, сутності відповідають окремим таблицям, а об'єкти визначають записи в цих таблицях.

Ще одним ключовим компонентом діаграми є зв'язки, які вказують на логічні залежності між даними. Розрізняють різні типи зв'язків, такі як один до одного, один до багатьох та багато до багатьох. Зв'язок багато до багатьох реалізується лише на рівні логічної моделі (рис. 3.1.1).

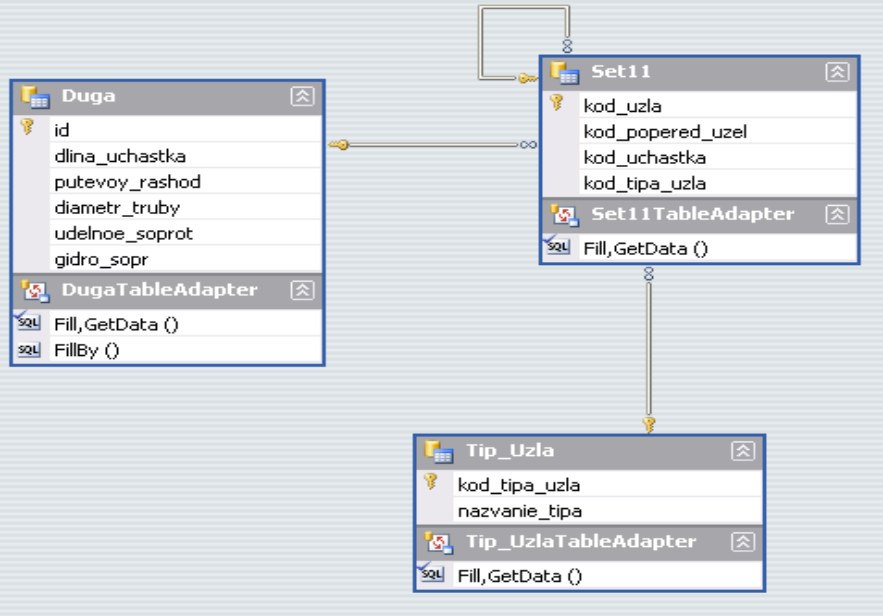


Рис. 3.1.1

У процесі створення логічної моделі обирається рівень зі списку, розташованого у правій частині інструментальної панелі ERWin. Зв'язки формуються між двома сутностями (об'єктами). На рівні логічної моделі діаграми виділяються зв'язки "один до багатьох" (ідентифікований та неідентифікований) і "багато до багатьох". На рівні фізичної моделі зв'язок "багато до багатьох" автоматично перетворюється у декілька зв'язків "один до багатьох".

На рисунку 3.1.2 представлено структуру, яка повинна відтворювати мережу. Entity Editor - це діалогове вікно, в якому визначаються характеристики об'єкту. Його можна викликати, натискавши праву клавішу миші, коли курсор розташований на зображенні об'єкту. В даному вікні визначають назву сутності та її характеристики. Кожну сутність слід описати текстово в закладці "Definition". Закладки "Note", "Note2", "Note3" та "UPD" (User Defined Properties - визначені користувачем якості) використовуються для додаткових коментарів і ознак, які стосуються сутності та конкретного екземпляру сутності - об'єкту.

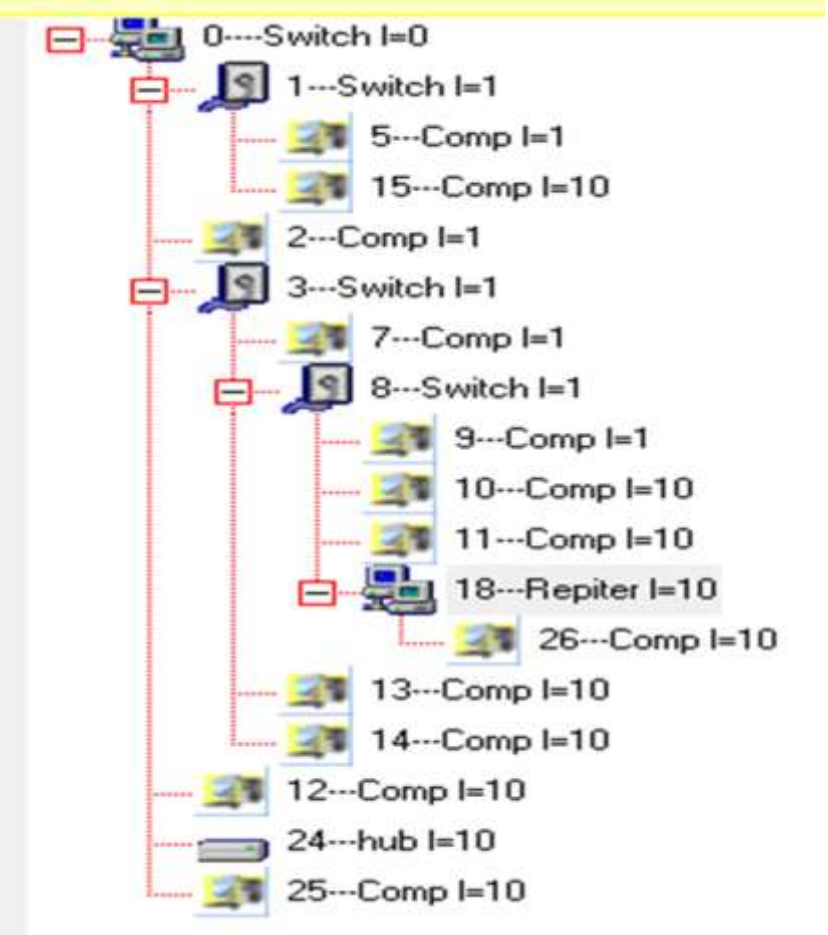


Рис. 3.1.2

Закладка "Note" дозволяє додати додаткові зауваження до опису об'єкту.

Закладка "Note2" дозволяє зазначити важливі запитання чи відомості, що стосуються об’єкту, який описується.

Закладка "Note3" призначена для введення прикладів даних для розглянутої сутності.

Атрибут відтворює визначену характеристику об'єкта. У реляційній базі даних атрибутом є колонка таблиці. Характеристики атрибута включають його ім'я, тип і область визначення. Атрибут може мати статус ключового або не ключового, а також може дозволяти чи забороняти значення NULL.

Ключем, або ключовим атрибутом, називають атрибут, який служить унікальним ідентифікатором запису. В таблиці може бути кілька ключів, з яких виділяють первинний ключ та інші.

Для створення переліку атрибутів для певного об'єкту слід натискати праву клавішу миші, коли її курсор розташований на зображенні цього об'єкту. Це спричинить виведення діалогового вікна "Attribute Editor", де ви встановите назви та характеристики атрибутів. Закладка "Definition" використовується для визначення атрибутів. Встановлення первинного ключа здійснюється в розділі "General" шляхом встановлення відповідного прапорця.

Область визначення атрибута отримала назву "домен". Домен - це набір значень, з якого обирається конкретне значення для атрибуту. Кожен атрибут може визначатися лише в рамках одного домену, проте для одного домену може бути визначено кілька атрибутів. До домену входять не лише тип даних, а й область існування даних. Домен визначається лише один раз і використовується у логічній та фізичній моделі.

Використання доменів спрощує взаємодію з даними для розробників на етапі проектування і для адміністраторів баз даних під час експлуатації системи. Логічний рівень дозволяє описувати домени без вказання конкретних фізичних характеристик. На фізичному рівні вони автоматично отримують конкретні характеристики, які є специфічними для обраної системи керування базами даних і можуть бути змінені вручну.

Для створення домену в логічній моделі використовується Domain Dictionary Editor. Його викликають з меню Edit/Domain Dictionary, натисканням на кнопку, розташовану у верхній лівій частині вікна Attribute Editor.

ERWin надає інструмент, який у значній мірі спрощує процес створення нових атрибутів у моделі за допомогою використання доменів. Викликати і закривати це вікно можна, натискавши гарячу клавішу CTRL+B.

Атрибут або група атрибутів, які унікально ідентифікують сутність, є первинним ключем. На діаграмі атрибути первинного ключа відображаються в списку атрибутів вище горизонтальної лінії. На діаграмі об'єкт-зв’язок ці

атрибути відсутні, і вперше вони з'являються на діаграмі, що базується на ключах (Key Based model).

Обрання первинного ключа є завданням, що потребує уважного вивчення, оскільки якість цього вибору може впливати на ефективність всієї системи в цілому. У багатьох випадках кілька атрибутів можуть претендувати на роль первинного ключа і вони називаються потенційними ключами (Candidate keys).

Складні ключі включають в себе декілька атрибутів, які розміщуються вище горизонтальної лінії на діаграмі. Щоб потенційний ключ став первинним, він повинен відповідати наступним критеріям:

- Вимога "бути унікальним" означає, що жодні два записи в базі даних не можуть мати ідентичні значення для визначеного первинного ключа. Кожен запис повинен мати унікальний ідентифікатор, який визначає його в межах бази даних.
- Вимога "бути компактним" для складного ключа означає, що всі атрибути, які входять до складного ключа, мають значущий внесок у визначення його унікальності. Жоден атрибут не може бути вилучений з ключа без порушення унікальності ключа. В інших словах, кожен атрибут у складному ключі повинен бути суттєвим для ідентифікації запису в базі даних.
- Вимога "первинний ключ не може мати значення NULL" означає, що для кожного запису в таблиці значення первинного ключа повинно бути унікальним і не може бути визначено як NULL. Кожен запис має обов'язковий та унікальний ідентифікатор, який не може бути порожнім.
- Вимога "значення атрибутів первинного ключа не змінюються протягом усього існування системи (або сутності)" означає, що раз визначений первинний ключ для запису в таблиці, його значення не може бути змінене протягом усього періоду функціонування системи чи об'єкта, якому він служить ідентифікатором. Це сприяє стабільності унікального

ідентифікатора і уникненню ситуацій, коли значення ключа втрачає свою унікальність через зміну.

Ключі, які не були обрані як первинні, можуть використовуватися як альтернативні ключі для індексації бази даних за необхідності. Інколи відповідно до бізнес-правил обирають атрибут для індексації, який не є унікальним для кожного запису, наприклад, прізвище клієнта. Цей атрибут відомий як інверсний вхід і позначається на логічній схемі. Первинний ключ обирається із списку атрибутів і може бути встановлений за допомогою вкладки "General" у діалоговому вікні Attribute Editor.

Атрибут, обраний як первинний ключ, можна перетягнути за допомогою миші до області первинного ключа, що розташована в списку атрибутів нижче горизонтальної лінії. Об'єкти зв'язані між собою логічним відношенням. На рівні логічної моделі можна задати ідентифікований зв'язок один-до-багатьох, багато-до-багатьох, а також неідентифікований зв'язок один-до-багатьох.

Ідентифікований зв'язок визначає зв'язок між батьківською (незалежною) та дочірньою (залежною) сутностями. Атрибути первинного ключа батьківської сутності автоматично стають атрибутами первинного ключа дочірньої сутності. У пакеті ERWin ідентифікований зв'язок відображається цільною лінією з товстою крапкою на дочірньому кінці зв'язку на діаграмі. Коли між сутностями існує ідентифікований зв'язок, дочірня сутність стає залежною від батьківської сутності. Це означає, що неможливо внести довільні зміни до дочірньої сутності; можна змінювати її лише відповідно до конкретних правил, щоб забезпечити цілісність даних між незалежною та залежною сутностями.

Сутність, що є залежною, відображається у вигляді прямокутника з округленими кутами. При неідентифікованому зв'язку дочірня сутність залишається незалежною, і атрибути первинного ключа батьківської сутності стають частиною не ключових компонентів батьківської сутності. Неідентифікований зв'язок використовується для з'єднання незалежних

об'єктів. Наприклад, об'єкт "Вершина" може існувати незалежно від будь-якого об'єкта "Дуга". На діаграмі зв'язок може бути позначений пунктирною лінією, що вказує на його неідентифікований характер.

Усі зв'язки повинні мати назву, яка включає дієслово або фразу з дієсловом (Relationship Verb Phrases). Це ім'я відображає певні обмеження або бізнес-правила, пов'язані з даним зв'язком.

У разі існування зв'язку між сутностями автоматично генеруються зовнішні ключі. Атрибути первинного ключа батьківської сутності автоматично копіюються до поля зовнішнього ключа в дочірній сутності і позначаються (FK) після свого імені. Якщо підлегла сутність отримує однаковий зовнішній ключ від кількох батьківських сутностей, для уникнення дублювання атрибутів з однаковими іменами відбувається уніфікація ключа, що означає його відображення на діаграмі лише один раз. Зовнішній ключ включається до поля зовнішнього ключа при ідентифікованому зв'язку і до загального списку атрибутів у разі неідентифікованого зв'язку.

Рекурсивний зв'язок виникає між екземплярами однієї сутності. Існують два основних типи рекурсивного зв'язку: ієрархічний та мережевий.

Ієрархічний зв'язок вказує на існування залежності "предок–потомок" між записами однієї сутності. Цей зв'язок може бути тільки неідентифікованим, оскільки ієрархічна залежність існує лише між окремими записами. У зв'язку з цим у дочірньому записі відбувається включення ключового атрибута батьківського запису у вигляді зовнішнього ключа у дочірній запис, що переміщується до області не ключових атрибутів(рис. 3.1.3).

Рис. 3.1.3

Другий варіант рекурсії - це мережевий, який відображає взаємозалежність "багато до багатьох" між конкретними екземплярами однієї сутності. Для втілення цього зв'язку можна використовувати створення додаткової сутності. Наприклад, у базі даних перепису населення можуть зберігатися дані про членів кожної сім'ї та їхні родинні зв'язки. Така ситуація може виникнути, коли одна особа, наприклад, Петренко, виступає як батько і син щодо інших членів родини. Для вирішення цієї ситуації потрібно створити додаткову таблицю, в якій будуть визначені різні типи відносин між членами кожної сім'ї, які можна інтерпретувати як їхні ролі. Ім'я ролі широко використовується для втілення рекурсивних зв'язків, тобто присвоюється атрибутам при створенні зв'язків між об'єктами у таких ситуаціях:

- при існуванні кількох аналогічнних зв'язків між двома сутностями;
- при існуванні рекурсивного зв'язку між екземплярами однієї сутності.

Зв'язок "багато до багатьох" може бути заданий лише на рівні логічної моделі даних і позначається як безперервна лінія з двома крапками на кінцях. Коли переходимо до фізичного рівня, ERWin автоматично трансформує цей зв'язок, створюючи нову таблицю та встановлюючи два нових зв'язки "один до багатьох" від попередніх до нової таблиці. При цьому назва нової таблиці формується автоматично як "Ім'я 1" & "Ім'я 2".

3.2. Інструкція користувача

Після запуску інформаційно-довідкової системи з’явиться вікно завантаження програми(рис. 3.2.1).

Рис. 3.2.1

Як тільки завершиться завантаження, ви побачите головне вікно системи(рис. 3.2.2).

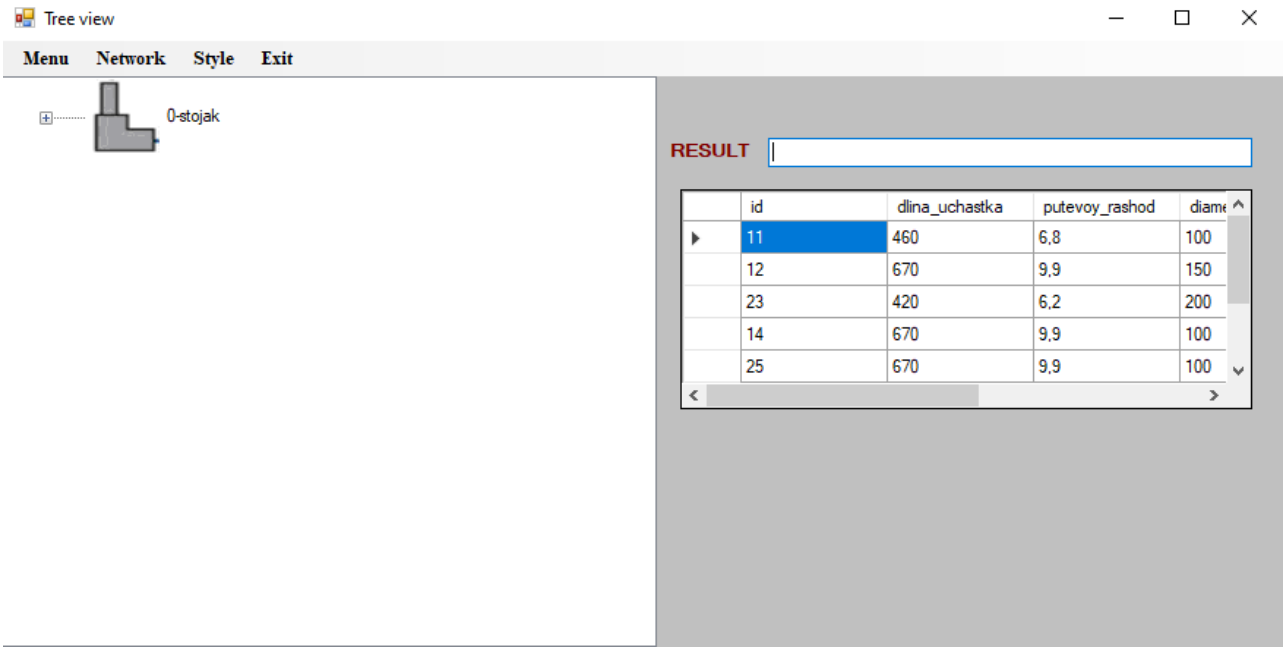


Рис. 3.2.2

У цьому вікні ви можете побачити дерево рішень нашого проекту, яке встановлене за замовчуванням(рис. 3.2.3).

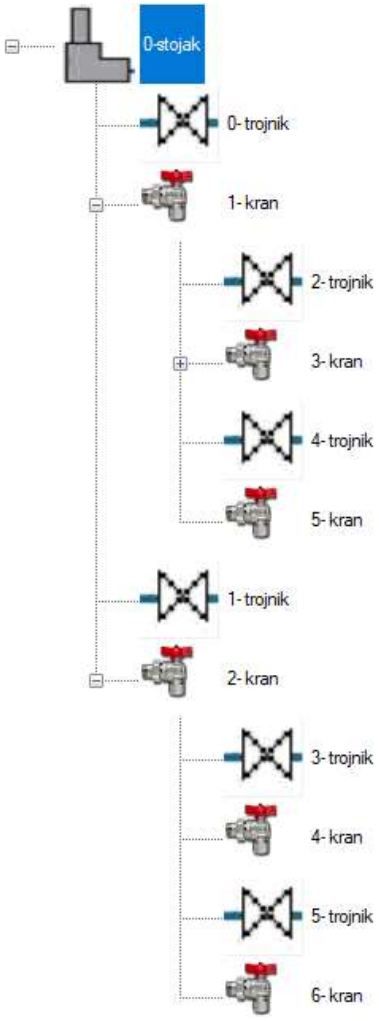


Рис. 3.2.3

Якщо вам не підходить елемент дерева рішень ви можете видалити його за допомогою меню. Для цього потрібно виділити елемент і вибрати відповідний пункт меню(рис. 3.2.4).

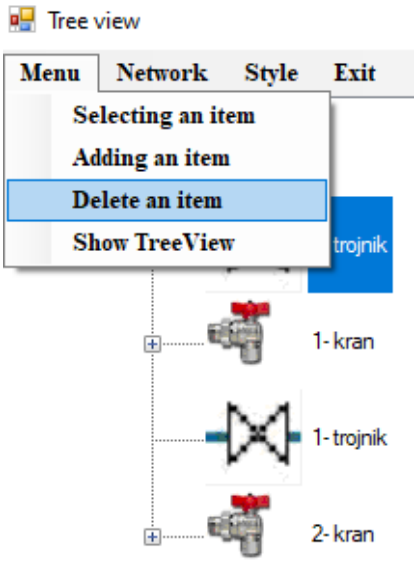


Рис. 3.2.4

Для додання нового елемента необхідно вибрати батьківський елемент і вибрати в меню пункт Selecting an item і натиснути на кнопку Choice у вікні, що з’явиться. Далі необхідно вибрати пункт Adding an item і обравши елемент натиснути кнопку Choice(рис. 3.2.5 – 3.2.7).

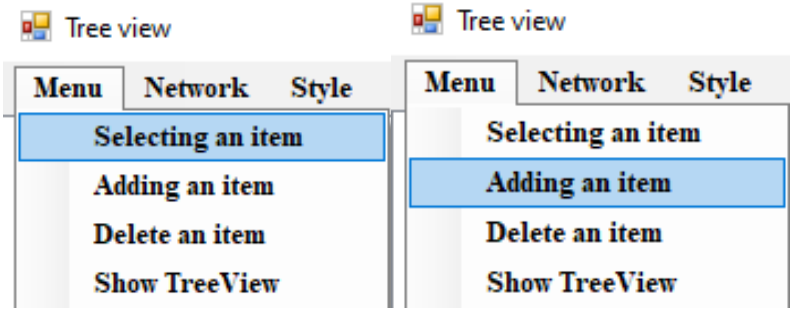


Рис. 3.2.5

Рис. 3.2.6

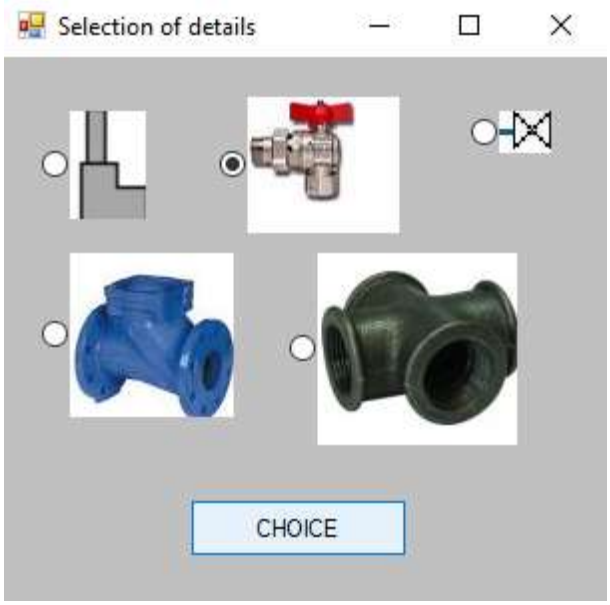


Рис. 3.2.7

Для того, щоб зберегти дерево рішень, яке знаходиться в межах вікна необхідно вибрати пункт меню Show Tree View. У вікні, що ви після цього побачите натисніть на Save і виберіть формат, в якому його слід зберегти. Для закриття вікна натисніть Close(рис. 3.2.8 – 3.2.9).

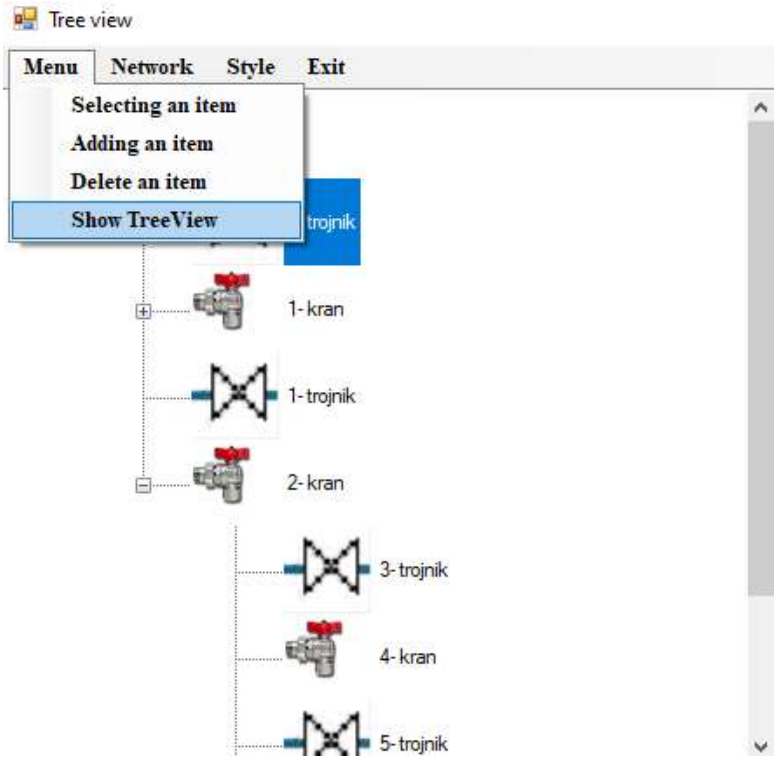


Рис. 3.2.8

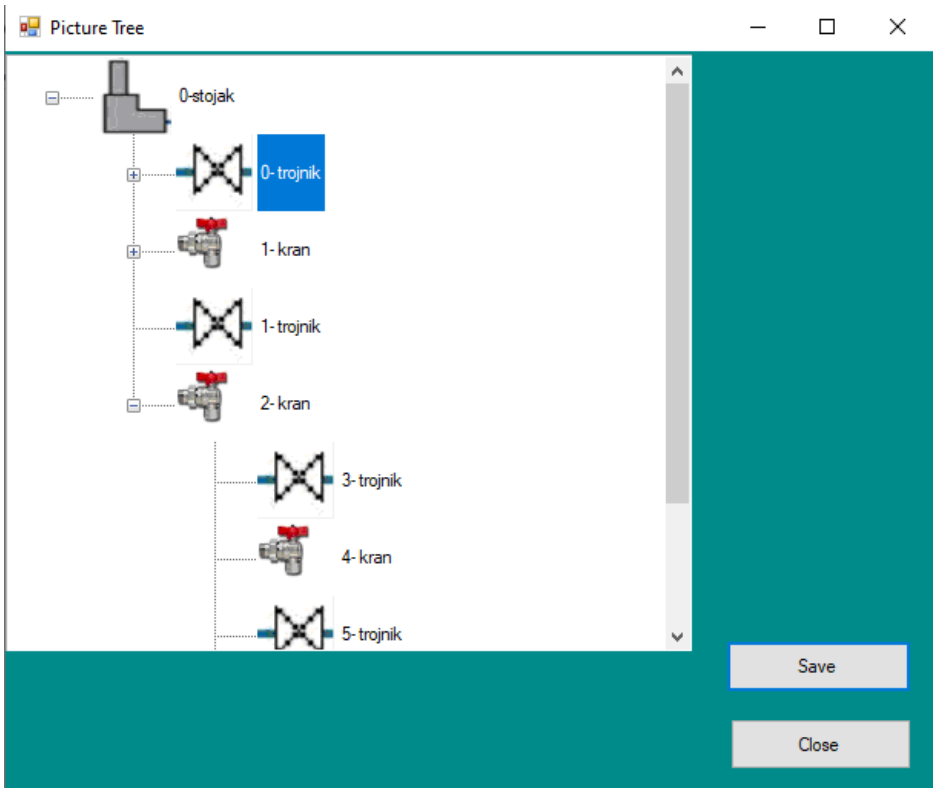


Рис. 3.2.9

Також можна змінити стиль відображення дерева рішень: стерти лінії та кнопки згортання/розгортання(рис. 3.2.10).

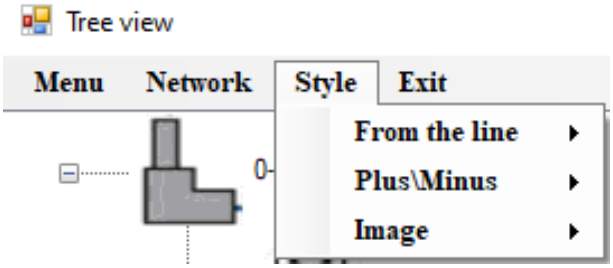


Рис. 3.2.10

У вікні можна побачити інформацію про вибраний елемент(рис. 3.2.11).

Рис. 3.2.11

Для аналізу мережі потрібно вибрати відповідний пункт у меню. Після завершення аналізу натиснути ОК у спливаючому вікні. Інформацію буде виведено у полі(рис. 3.2.12 – 3.2.13).

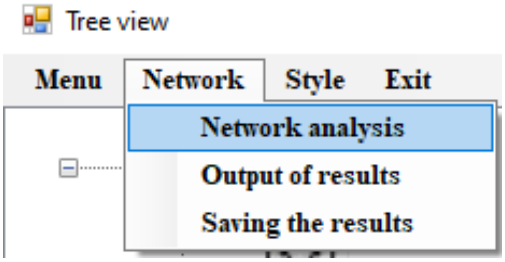


Рис. 3.2.12

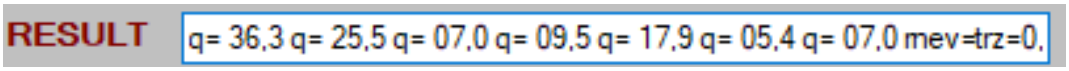


Рис. 3.2.13

Натиснувши на пункти Output of results і Saving the results можна відповідно вивести результати з фалу та зберегти їх у файл.

Для виходу з інформаційно-довідкової системи потрібно вибрати пункт меню Exit(рис. 3.2.13).

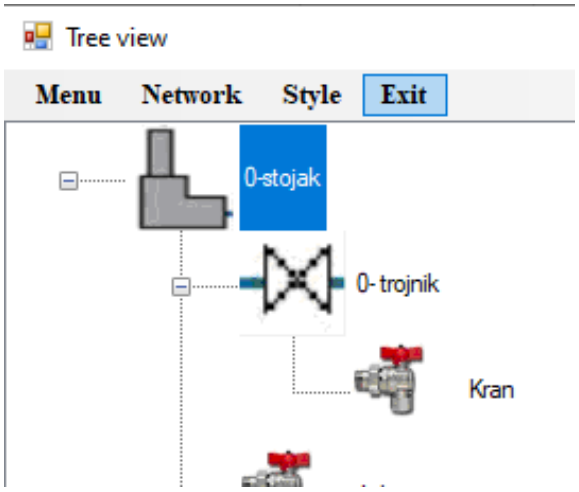


Рис. 3.2.13

3.3. Висновок до розділу

У даному відділі ми обґрунтовуємо корисність методів втілення програмного продукту відповідно до узгоджень ERWin - MVC за допомогою шаблонів Microsoft Visual Studio MFP.

Крім того, ми розглядаємо інструкцію користувача, яка є достатньо докладною і дозволяє кожному легко розпочати роботу з програмою.

Наші структурні інформаційні моделі, схеми даних і моделі для представлення та зберігання інформації, а також архітектура бази даних відповідають вимогам поставленої задачі. Реалізація програмних рішень

системи спроектована і виконана з використанням типових, зрозумілих, уніфікованих і стандартизованих елементів, що сприяє зручності їхнього розуміння та використання.

ВИСНОВКИ

У результаті виконання дипломної магістерської роботи були отримані наступні результати:

Здійснено аналіз основних аспектів гідравлічних мереж.

Визначено основне призначення інформаційно-довідкової системи для дослідження та аналізу ефективності інженерних мереж підприємства. Обрано середовище розробки для створення системи.

Розроблено інструкцію користувача для полегшення роботи з інформаційно-довідковою системою.

Програма відповідає технічним вимогам та завданням і надає результати у зрозумілому та наочному вигляді.

Система має сучасний дизайн та інтуїтивно-зрозумілий інтерфейс, що дозволяє користувачам з різним рівнем навичок отримувати необхідну інформацію. Проектування системи використовує максимальну кількість стандартних елементів, що сприяє зручності їхнього розуміння та використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Барвінський А. Ф., Олексів І. Я., Крупка З. І. та ін. Математичне програмування. Дослідження операцій: навч. посіб. для студ. вищ. навч. закл. Львів: Інтелект-Захід, 2008. 468 с.

2. Бейко І. В., Зінько П. М., Наконечний О. Г. Задачі, методи і алгоритми оптимізації : навч. посіб. Рівне : НУВГП, 2011. 624 с.

3. Богаєнко І. М., Григорків В. С., Бойчук М. В., Рюмшин М. О. Математичне програмування: навч. посіб. К.: Логос, 1996. 266 с.

4. Волонтир Л. О, Зелінська О. В., Потапова Н. А., Чіков І. А., Чисельні методи: навч. посіб. Вінниця: ВНАУ, 2020 322 с.

5. Глушик М. М., Копич І. М., Пенцак О. С., Сороківський В. М. Математичне програмування: підручник. Львів: "Новий світ", 2006. 214 с.

6. Жалдак М. І., Триус Ю. В. Основи теорії і методів оптимізації: навч. посіб. Черкаси: Брама-Україна, 2005. 608 с.

7. Жилинскас А. Г. Глобальная оптимизация. Аксиоматика статических моделей, алгоритмы, применения. –Вильнюс: Мокслас, 1986. – 166 с.

8. Корольов М. Є., Павленко В. І., Савіна О. В., Тимошенко А. Г. Дослідження операцій і методи оптимізації: навч. посіб. Київ: Університет "Україна", 2007. 177 с.

9. Лавров Є. А., Перхун Л. П., Шендрик В. В. та ін. Математичні методи дослідження операцій : підручник. Суми : Сумський державний університет, 2017. 212 с

10. Ладогубець Т. С., Фіногенов О. Д. Двоїстість в лінійному програмуванні: практикум з дисципліни «Методи оптимізації»[Електронний ресурс]: навч. посіб. для студ. спеціальності 113 «Прикладна математика», спеціалізації «Наука про дані та математичне моделювання». Київ: КПІ ім. Ігоря Сікорського, 2019. 59 с.

11. Мовчан А.П., Степанець О.В. Методи статичної оптимізації. навч. посіб. К.: НТУУ «КПІ», 2012. 138 с.

12. Моклячук, М. П. Варіаційне числення. Екстремальні задачі. Підручник. – К. : Видавничо-поліграфічний центр "Київський університет", 2009. 380 с.

13. Наконечний С. І., Савіна С. С. Математичне програмування: навч. посіб. КНЕУ, 2003. 452 с.

14. Нефьодов Ю. М., Балицька Т. Ю. Методи оптимізації в прикладах і задачах: навч. посіб. – К.: Кондор, 2011. 324 с.

15. Самойленко М. І. Математичне програмування: навч. посіб. Харків: Основа, 2001. 424 с

16. Толубко В. Б., Беркман Л. Н. Методи оптимізації: підручник. ДУТ, 2016. 442 с.

Додаток А

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void timer1_Tick(object sender, EventArgs e)
      {
         if (progressBar1.Value < progressBar1.Maximum)
         {
            progressBar1.Value += progressBar1.Step;
         } else
         {
            Form form2 = new Form2();
            form2.Show();
            this.Hide();
            timer1.Enabled = false;
         }

      }
   }
}



using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
using System.IO;

namespace WindowsFormsApplication1
{
   public partial class Form2 : Form
   {
      public Form2()
      {
         InitializeComponent();
      }

      Form form3 = new Form3();
      int stojakTag = 1;
      int kranTag = 7;
```

```
        int trojnikTag = 7;
        int ventilTag = 0;
        int zadvigkaTag = 0;

        private TreeNode inod(TreeNodeCollection mynodcol, String tg)
        {
            foreach (TreeNode rn in mynodcol)
            {
                if (rn.Tag.ToString() == tg)
                {
                    return rn;
                }
            }
            foreach (TreeNode rn in mynodcol)
            {
                if (inod(rn.Nodes, tg) != null)
                {
                    return inod(rn.Nodes, tg);
                }
            }
            return null;
        }

        private TreeNode inod2(TreeNodeCollection mynodcol, String tg, OleDbConnection con)
        {
            OleDbCommand com1 = new OleDbCommand();
            com1.Connection = con;
            com1.CommandType = CommandType.Text;

            foreach (TreeNode rn in mynodcol)
            {
                com1.CommandText = "insert into set2 ( kod_uzla, kod_popered_uzel, kod_uchastka )" + "values (" +
rn.Tag.ToString() +
                    ", " + tg + ", 7);";
                com1.ExecuteNonQuery();
                return inod2(rn.Nodes, rn.Tag.ToString(), con);
            }
            return null;
        }

        private void inodlin(TreeNodeCollection mynodcol, String tg, ref int nomogr, ref analiz a, ref int nper)
        {
            nper++;
            if (mynodcol.Count == 0)
            {
                return;
            }
            a.matlin[nomogr, Int32.Parse(tg)] = 1;
            foreach (TreeNode rn in mynodcol)
            {
                a.matlin[nomogr, Int32.Parse(rn.Tag.ToString())] = -1;
            }
            nomogr++;
            foreach (TreeNode rn in mynodcol)
            {
                inodlin(rn.Nodes, rn.Tag.ToString(), ref nomogr, ref a, ref nper);
            }
        }

        private void inodqrt(TreeNodeCollection mynodcol, String tg, ref int nomogr, ref analiz a)
        {
            TreeNode temp;
```

```
            if (mynodcol.Count == 0)
            {
                a.matqrt[nomogr, Int32.Parse(tg)] = ksopr(Int32.Parse(tg));
                a.matqrt[nomogr, 0] = pdav(Int32.Parse(tg));
                nomogr++;
                return;
            }
            foreach (TreeNode rn in mynodcol)
            {
                if (nomogr == 10)
                {
                    return;
                }
                temp = rn.Parent;
                while (temp != null)
                {
                    a.matqrt[nomogr, Int32.Parse(temp.Tag.ToString())] = ksopr(Int32.Parse(tg));
                    temp = temp.Parent;
                }
                a.matqrt[nomogr, Int32.Parse(tg)] = ksopr(Int32.Parse(tg));

                inodqrt(rn.Nodes, rn.Tag.ToString(), ref nomogr, ref a);
            }
        }

        private double ksopr(int n)
        {
            // гідравлічний опір дуги
            return 2;
        }

        private double pdav(int n)
        {
            // перепад тиску між входом і вершиною
            return 5;
        }

        private void Form2_Load(object sender, EventArgs e)
        {
            this.dugaTableAdapter.Fill(this.delo_trubaDataSet1.Duga);
            OleDbConnection connection = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=delo_truba.mdb");
            OleDbCommand com = new OleDbCommand("select * from set1");
            com.CommandType = CommandType.Text;
            OleDbDataReader dr;
            OleDbDataReader dr1;
            connection.Open();
            com.Connection = connection;
            dr = com.ExecuteReader();

            TreeNode pnode = new TreeNode();
            TreeNode cnode = new TreeNode();
            TreeNode nkran = new TreeNode("kran");
            nkran = new TreeNode();
            string s;
            string s2 = "";
            string key;

            while (dr.Read())
            {
                if (dr.GetValue(0).ToString() == "0")
                {
```

```
            tv1.BeginUpdate();
            pnode = tv1.Nodes.Add(dr.GetValue(0).ToString() + "-" + "stojak");
            pnode.Tag = dr.GetValue(0).ToString();
            pnode.ImageIndex = 5;
            pnode.SelectedImageIndex = 5;
            tv1.EndUpdate();
         }
         else
         {
            s2 = "child " + dr.GetValue(0).ToString();
            key = dr.GetValue(0).ToString();
            cnode = pnode.Nodes.Add(key, s2);
            cnode.Tag = key;
            cnode.Text = key + "- kran";
         }
         s = dr.GetValue(1).ToString();
         pnode = inod(tv1.Nodes, s);
         if (pnode == null)
         {
            return;
         }
         key = dr.GetValue(0).ToString();
         cnode = pnode.Nodes.Add(key, s2);
         cnode.Tag = key;
         cnode.Text = key + "- trojnik";

         cnode.ImageIndex = 4;
         cnode.SelectedImageIndex = 4;

         if (dr.GetValue(4).ToString() == "kran")
         {
            cnode.ImageIndex = 2;
            cnode.SelectedImageIndex = 2;

            pnode = cnode;
            cnode.ExpandAll();
         }
      }

      com.CommandText = "select duga.dlina_uchastka, duga.putevoy_rashod, duga.diametr_truby,
duga.udelnoe_soprot," +
         " duga.gidro_sopr, [duga]![gidro_sopr]*([duga]![putevoy_rashod]*[duga]![putevoy_rashod]) as
выражение1 from duga;";
      dr.Close();
      dr1 = com.ExecuteReader();

      OleDbCommand com2 = new OleDbCommand("delete set2.* from set2");

      com2.CommandType = CommandType.Text;
      com2.Connection = connection;
      com2.ExecuteNonQuery();

      pnode = inod2(tv1.Nodes, "0", connection);
      connection.Close();
   }

   private void tv1_AfterSelect(object sender, TreeViewEventArgs e)
   {
      label1.Text = "text: " + tv1.SelectedNode.Text +
         " tag: " + tv1.SelectedNode.Tag + " index: " +
         tv1.SelectedNode.Index.ToString() + " childs: " +
         tv1.SelectedNode.GetNodeCount(true).ToString() +
```

```
        " count= " + tv1.Nodes.Count.ToString() +
        " childcount= " + tv1.SelectedNode.Nodes.Count;
}

private void selectingAnItemToolStripMenuItem_Click(object sender, EventArgs e)
{
    form3.Show();
}

private void addingAnItemToolStripMenuItem_Click(object sender, EventArgs e)
{
    TreeNode cNode = new TreeNode();
    TreeNode nNode = new TreeNode();

    form3.ShowDialog();
    cNode = tv1.SelectedNode;
    nNode = cNode.Nodes.Add("New node");

   if (RadioCheck.RadioCheckedHandler("radioButton1"))
   {
      nNode.ImageIndex = 5;
      nNode.SelectedImageIndex = 5;
      nNode.Text = "Stojak";
      nNode.Tag = stojakTag;
      stojakTag++;
   }

   if (RadioCheck.RadioCheckedHandler("radioButton2"))
   {
      nNode.ImageIndex = 0;
      nNode.SelectedImageIndex = 0;
      nNode.Text = "Kran";
      nNode.Tag = kranTag;
      kranTag++;
   }

   if (RadioCheck.RadioCheckedHandler("radioButton3"))
   {
      nNode.ImageIndex = 4;
      nNode.SelectedImageIndex = 4;
      nNode.Text = "Trojnik";
      nNode.Tag = trojnikTag;
      trojnikTag++;
   }

   if (RadioCheck.RadioCheckedHandler("radioButton4"))
   {
      nNode.ImageIndex = 1;
      nNode.SelectedImageIndex = 1;
      nNode.Text = "Zapornij ventil";
      nNode.Tag = ventilTag;
      ventilTag++;
   }

   if (RadioCheck.RadioCheckedHandler("radioButton5"))
   {
      nNode.ImageIndex = 2;
      nNode.SelectedImageIndex = 2;
      nNode.Text = "Zadvigka";
      nNode.Tag = zadvigkaTag;
      zadvigkaTag++;
   }
```

```
}

private void deleteAnItemToolStripMenuItem_Click(object sender, EventArgs e)
{
   if (tv1.SelectedNode != null)
   {
      tv1.SelectedNode.Remove();
   }
}

private void showTreeViewToolStripMenuItem_Click(object sender, EventArgs e)
{
   Bitmap bm = new Bitmap(tv1.Width, tv1.Height);
   Rectangle r = new Rectangle();
   r.X = 1;
   r.Y = 1;

   r.Width = tv1.Width;
   r.Height = tv1.Height;

   tv1.DrawToBitmap(bm, r);

   Form4 form4 = new Form4();
   form4.setBoxInfo(tv1.Width, tv1.Height, bm);
   form4.Show();
}

private void networkAnalysisToolStripMenuItem_Click(object sender, EventArgs e)
{
   analiz a = new analiz();
   a.matlin = new double[10, 27];
   a.matqrt = new double[10, 27];
   int nogr = 0;
   int nper = 0;
   inodlin(tv1.Nodes[0].Nodes, tv1.Nodes[0].Nodes[0].Tag.ToString(), ref nogr, ref a, ref nper);
   a.nlinogr = nogr - 1;
   nogr = 1;
   inodqrt(tv1.Nodes, tv1.Nodes[0].Nodes[0].Tag.ToString(), ref nogr, ref a);
   a.nper = nper - 1;
   a.nsqrogr = nogr - 1;
   a.trz = 0.1;
   a.solutionset();
   MessageBox.Show("Ув'язування мережі завершено!!!");
}

private void outputOfResultsToolStripMenuItem_Click(object sender, EventArgs e)
{
   String winpath;
   String s;
   String fname;
   StreamReader sr;
   winpath = System.Environment.GetEnvironmentVariable("D:/Windows");
   if (winpath == " ")
   {
      return;
   }
   fname = winpath + "9999.txt";
   if (File.Exists(fname))
   {
      try
      {
         sr = File.OpenText(fname);
```

```
            s = sr.ReadToEnd();
            sr.Close();
            textBox1.Text = s;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return;
        }
    }
    else
    {
        MessageBox.Show("не знайдено потрібний файл!!! " + fname);
        return;
    }
}

private void savingTheResultsToolStripMenuItem_Click(object sender, EventArgs e)
{
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.Filter = "txt files (*.Txt)| *.txt| all Files (*.*)|*.*";
    sfd.FilterIndex = 2;
    sfd.RestoreDirectory = true;
    if (sfd.ShowDialog() == DialogResult.OK)
    {
        FileInfo f = new FileInfo(sfd.FileName);
        StreamWriter sv = f.CreateText();
        if (sv != null)
        {
            sv.Write(textBox1.Text);
            sv.Close();
        }
    }
}

private void plusToolStripMenuItem_Click(object sender, EventArgs e)
{
    tv1.ShowLines = true;
}

private void minusToolStripMenuItem_Click(object sender, EventArgs e)
{
    tv1.ShowLines = false;
}

private void plusToolStripMenuItem1_Click(object sender, EventArgs e)
{
    tv1.ShowPlusMinus = true;
}

private void minusToolStripMenuItem1_Click(object sender, EventArgs e)
{
    tv1.ShowPlusMinus = false;
}

private void image1ToolStripMenuItem_Click(object sender, EventArgs e)
{
    tv1.ImageList = iml1;
}

private void image2ToolStripMenuItem_Click(object sender, EventArgs e)
{
```

```
            tv1.ImageList = iml2;
        }

        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form3 : Form
    {
        public Form3()
        {
            RadioCheck.RadioCheckedHandler = new RadioCheck.RadioChecked(this.GetChecked);
            InitializeComponent();
        }

        public bool GetChecked(String name)
        {
            switch (name)
            {
                case "radioButton1":
                    return radioButton1.Checked;
                case "radioButton2":
                    return radioButton2.Checked;
                case "radioButton3":
                    return radioButton3.Checked;
                case "radioButton4":
                    return radioButton4.Checked;
                case "radioButton5":
                    return radioButton5.Checked;
            }
            return radioButton1.Checked;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.Hide();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.IO;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Printing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
   public partial class Form4 : Form
   {
      public Form4()
      {
         InitializeComponent();
      }

      public void setBoxInfo(int width, int height, Bitmap bm)
      {
         pictureBox1.Width = width;
         pictureBox1.Height = height;
         pictureBox1.BackgroundImage = bm;
      }

      private void button1_Click(object sender, EventArgs e)
      {
         if (pictureBox1.BackgroundImage != null)
         {
            SaveFileDialog savedialog = new SaveFileDialog();
            savedialog.Title = "Сохранить картинку как...";
            savedialog.OverwritePrompt = true;
            savedialog.CheckPathExists = true;
            savedialog.Filter = "Image Files(*.BMP)|*.BMP|Image Files(*.JPG)|*.JPG|Image
Files(*.GIF)|*.GIF|Image Files(*.PNG)|*.PNG|All files (*.*)|*.*";
            savedialog.ShowHelp = true;
            if (savedialog.ShowDialog() == DialogResult.OK)
            {
               try
               {
                  pictureBox1.BackgroundImage.Save(savedialog.FileName,
System.Drawing.Imaging.ImageFormat.Png);
               }
               catch
               {
                  MessageBox.Show("Невозможно сохранить изображение", "Ошибка",
                  MessageBoxButtons.OK, MessageBoxIcon.Error);
               }
            }
         }
      }

      private void button2_Click(object sender, EventArgs e)
      {
         this.Close();
      }
   }
}



using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;

namespace WindowsFormsApplication1
{
   class analiz
   {
      public double[,] matlin;
      public double[,] matqrt;
      public double[] x;
      public int nlinogr;
      public int nsqrogr;
      public int nper;
      public double trz;

      public analiz()
      {

      }

      public analiz(int m, int n)
      {
        double[,] matlin = new double [m, n];
        double[,] matqrt = new double [m, n];
      }

      public double fnevjazlin(int j)
      {
         double rnev = 0, norm = 0, t;
         for (int i = 0; i < nper; i++)
         {
            rnev += matlin[j, i] * x[i];
            norm += matlin[j, i] * matlin[j, i];
         }
         rnev = matlin[j, 0] - rnev;
         if (Math.Abs(rnev) <= trz)
         {
            return rnev;
         }
         t = rnev / norm;

         for (int i = 0; i < nper; i++)
         {
            x[i] += matlin[j, i] * t;
         }
         return rnev;
      }

      public double fnevjazqrt(int j)
      {
         double rnev = 0, norm = 0, t;
         for (int i = 0; i < nper; i++)
         {
            rnev += matqrt[j, i] * x[i] * x[i];
            norm += matqrt[j, i] * matqrt[j, i];
         }
         rnev = matqrt[j, 0] - rnev;
         if (Math.Abs(rnev) <= trz)
         {
            return rnev;
         }
         t = rnev / norm;
```

```
            double tr = 0;
            for (int i = 0; i < nper; i++)
            {
                if (t < 0)
                {
                    tr = -x[i] * x[i] / matqrt[j, i];
                    if (tr > t)
                    {
                        t = tr;
                    }
                }
            }
            for (int i = 0; i < nper; i++)
            {
                x[i] = Math.Sqrt(x[i] * x[i] + matqrt[j, i] * t);
            }
            return rnev;
        }

        public void solutionset()
        {
            this.x = new double[nper];
            int k = 0;
            for (int i = 0; i < nper; i++)
            {
                x[i] = 0;
            }
            double nevjaz = 99999;
            while ((nevjaz > trz) && (k <= 1000))
            {
                nevjaz = 0;
                k++;
                double rnev = 0;
                for (int j = 0; j <= nsqrogr; j++)
                {
                    rnev = fnevjazqrt(j);
                    if (Math.Abs(rnev) > nevjaz)
                    {
                        nevjaz = Math.Abs(rnev);
                    }
                }
                for (int j = 0; j <= nlinogr; j++)
                {
                    rnev = fnevjazlin(j);
                    if (Math.Abs(rnev) > nevjaz)
                    {
                        nevjaz = Math.Abs(rnev);
                    }
                }
            }
        }
    }
}
```