**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**
**Факультет прикладної математики**
**Кафедра програмного забезпечення комп'ютерних систем**

«На правах рукопису»
УДК 519.688

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2024 р.

# Магістерська дисертація

**на здобуття ступеня магістра за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»**
**зі спеціальності 121 Інженерія програмного забезпечення**
**на тему: «Спосіб та програмне забезпечення для організації та мінімізації часу виконання задач»**

Виконав:
Студент ІІ курсу, групи КП-21мп
Коваль Андрій Олександрович _____

Керівник:
Старший викладач кафедри ПЗКС, к.т.н.,
Хіцко Яна Володимирівна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри СПіСКС, к.т.н., доцент
Боярінова Юлія Євгенівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2024 року

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

Рівень вищої освіти – другий (магістерський)

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___»_____ 2022 р.

# ЗАВДАННЯ

## на магістерську дисертацію студенту

Ковалю Андрію Олександровичу

1. Тема дисертації «Спосіб та програмне забезпечення для організації та мінімізації часу виконання задач», науковий керівник дисертації Хіцко Яна Володимирівна, к.т.н., ст. викладач, затверджена наказом по університету від «09» листопада 2023 р. № 5217-С.
2. Термін подання студентом дисертації «15» січня 2024 р.
3. Об'єкт дослідження: процес планування порядку виконання задач та їх розподілення між працівниками з мінімізацією часу простою.
4. Предмет дослідження: методи, алгоритми та програмні засоби для оптимізації процесу планування та розподілу задач.
5. Перелік завдань, які потрібно розробити:
   - огляд існуючих рішень
   - проведення аналізу обраних алгоритмів;
   - розподілення задач за допомогою комбінованого алгоритму;
   - провести порівняння результатів роботи алгоритму з існуючими аналогами;
   - зробити висновок, основуючись на результатах дослідження
6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
   - блок схеми алгоритмів;
   - діаграми модулів ПЗ

- графіки часу роботи системи з певним розміром даних

7. Орієнтовний перелік публікацій:
   - Тези доповіді «Спосіб та програмне забезпечення для організації та мінімізації часу виконання задач» на конференції “Прикладна математика та ком’ютинг” (ПМК-2023)

8. Консультанти розділів дисертації

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| Нормоконтроль | Онай М.В., доцент кафедри ПЗКС | | |

9. Дата видачі завдання «10» жовтня 2022 р.

Календарний план

| № з/п | Назва етапів виконання магістерської дисертації | Термін виконання етапів магістерської дисертації | Примітка |
|---|---|---|---|
| 1. | Ґрунтовне ознайомлення з предметною галуззю | 16.12.2022 | |
| 2. | Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук | 02.02.2023 | |
| 3. | Робота над першим розділом магістерської дисертації; проведення наукового дослідження | 13.03.2023 | |
| 4. | Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення | 08.05.2023 | |
| 5. | Проведення наукового дослідження; робота над статтею за результатами наукового дослідження | 14.08.2023 | |
| 6. | Проведення наукового дослідження; робота над третім розділом магістерської дисертації | 19.09.2023 | |
| 7. | Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді ПМК-2023 | 09.11.2023 | |
| 8. | Оформлення текстової і графічної частини магістерської дисертації | 07.12.2023 | |

Студент _____ Андрій КОВАЛЬ

Науковий керівник дисертації  _____ Яна ХІЦКО

# РЕФЕРАТ

**Актуальність теми**. Актуальність даної теми полягає в зростаючій потребі в ефективному управлінні задачами в умовах стрімкого розвитку інформаційних технологій та збільшення обсягу даних. Розвиток методів для ефективного розподілу робіт між виконавцями є ключовим для підвищення продуктивності та конкурентоспроможності організацій.

Пропонований у цій роботі алгоритм, що інтегрує методи повного перебору та апроксимації Фогеля, разом з алгоритмом балансування навантаження, відкриває нові можливості для розв’язання цієї проблеми.

**Об’єкт дослідження**: процес планування порядку виконання задач та їх розподілення між працівниками з мінімізацією часу простою.

**Предмет дослідження**: методи, алгоритми та програмні засоби для оптимізації процесу планування та розподілу задач.

**Мета роботи**: розробка та валідація алгоритму, який зможе динамічно обирати підхід для оптимізації порядку виконання задач залежно від розміру набору даних.

**Методи досліджень**: Теоретичний та емпіричний аналіз існуючих методів і сервісів для розподілення задач.

**Наукова новизна** полягає у впровадженні алгоритму для розподілу задач, що складається з декількох різних за своєю сутністю етапів таких як балансування, алгоритм Фогеля та повного перебору.

**Практична цінність** отриманих в роботі результатів полягає у значному поліпшенні процесу розподілу задач. Розроблений алгоритм дозволяє оптимально розподіляти задачі між виконавцями, забезпечуючи ефективне використання ресурсів і мінімізацію часу виконання.

**Апробація роботи**. Основні положення і результати роботи представлені та обговорені на XVI науковій конференції магістрантів та

аспірантів «Прикладна математика та комп'ютинг» ПМК-2023 (м. Київ, 28-30 листопада 2023 р.)

**Структура та обсяг роботи.** Магістерська дисертація складається з вступу, п'яти основних розділів, висновків та додатків.

У вступі надано огляд дослідження, обґрунтовано актуальність теми, сформульовано мету і задачі, підкреслено наукову новизну та практичну значущість отриманих результатів, а також наведено інформацію про апробацію та впровадження результатів дослідження.

Перший розділ присвячено огляду існуючих методів та підходів у сфері розподілення завдань. Аналізується сильні та слабкі сторони цих методів і висвітлюється потреба у новому рішенні.

У другому розділі представлено розроблений алгоритм, який є модифікацією алгоритму для вирішення транспортної задачі.

Третій розділ охоплює технічні аспекти розробки алгоритму, включаючи вибір технологій та інструментів. Також тут описано процес програмної реалізації алгоритму та його практичне застосування.

Четвертий розділ містить детальний аналіз результатів, отриманих від реалізації алгоритму, із проведенням випробувань на різних наборах даних.

П'ятий розділ зосереджується на розробці бізнес-моделі для стартапу, включаючи аналіз ринку, виявлення зацікавлених сторін, формулювання унікальної ціннісної пропозиції, а також розрахунок та систематизацію доходів і витрат.

У висновках підбито підсумки проведеної роботи, висвітлено ключові здобутки та перспективи подальшого розвитку проекту.

Робота виконана на 75 аркушах, містить 3 додатки та посилання на список використаних літературних джерел з 17 найменувань. У роботі наведено 22 рисунки та 6 таблиць.

**Ключові слова**. Розподілення задач. Алгоритм апроксимації Фогеля. Алгоритми оптимізації.

# ABSTRACT

**Relevance of the topic**. The relevance of this topic lies in the growing need for effective task management in the context of the rapid development of information technology and the increase in data volume. The development of methods for efficiently distributing work among performers is key to increasing the productivity and competitiveness of organizations.

The algorithm proposed in this paper, which integrates the methods of full search and Vogel approximation, together with the load balancing algorithm, opens up new possibilities for solving this problem

**Object of research**: the process of planning the order of tasks and their distribution among employees with minimizing downtime.

**Subject of research**: methods, algorithms and software tools for optimizing the process of planning and distributing tasks.

**Purpose**: to develop and validate an algorithm that can dynamically select an approach to optimize the order of tasks depending on the size of the data set.

**Research methods**: Theoretical and empirical analysis of existing methods and services for task distribution.

**The scientific novelty** lies in the implementation of an algorithm for task allocation consisting of several different stages, such as balancing, Vogel's algorithm, and full search.

**The practical value** of the results obtained in this work is a significant improvement in the process of task distribution. The developed algorithm allows to optimally distribute tasks among performers, ensuring efficient use of resources and minimizing execution time.

**Approbation of work.** The main provisions and results of the work were presented and discussed at the XVI scientific conference of undergraduates and graduate students "Applied Mathematics and Computing" PMK-2023 (Kyiv, November 28-30, 2023).

**Structure and scope of the work**. The master's thesis consists of an introduction, five main chapters, conclusions and appendices.

The introduction provides an overview of the study, substantiates the relevance of the topic, formulates the goal and objectives, emphasizes the scientific novelty and practical significance of the results, and provides information on the testing and implementation of the study results.

The first section is devoted to an overview of existing methods and approaches in the field of task distribution. The strengths and weaknesses of these methods are analyzed and the need for a new solution is highlighted.

The second section presents the developed algorithm, which is a modification of the algorithm for solving the transportation problem.

The third section covers the technical aspects of the algorithm development, including the choice of technologies and tools. It also describes the process of software implementation of the algorithm and its practical application.

The fourth section contains a detailed analysis of the results obtained from the algorithm implementation, with tests on various data sets.

The fifth section focuses on the development of a business model for a startup, including market analysis, stakeholder identification, formulation of a unique value proposition, and calculation and systematization of income and expenses.

The conclusions summarize the results of the work done, highlighting the key achievements and prospects for further development of the project.

The work is executed on 75 pages, contains 3 appendices and references to a list of 17 references. The paper contains 22 figures and 6 tables.

**Key words**. Distribution of tasks. Vogel's approximation algorithm. Optimization algorithms.

**ЗМІСТ**

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ ..... 4
1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ДЛЯ ПЛАНУВАННЯ ПОРЯДКУ ВИКОНАННЯ ЗАДАЧ ТА ЇХ РОЗПОДІЛЕННЯ ..... 6
1.1. Аналіз основних методів планування порядку виконання задач ..... 6
1.2. Огляд існуючих рішень ..... 6
1.3. Основні метрики для аналізу ..... 11
1.4. Висновки до розділу ..... 19
2. МОДИФІКОВАНИЙ АЛГОРИТМ РОЗПОДІЛЕННЯ ЗАДАЧ ..... 22
2.1. Аргументація модифікованого алгоритму ..... 22
2.2. Модифікація алгоритмів для роботи зі списками задач та їх зв’язками ..... 22
2.3. Особливості модифікації алгоритму ..... 26
2.4. Висновки до розділу ..... 28
3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ АЛГОРИТМУ РОЗПОДІЛЕННЯ ЗАДАЧ В ЧАСІ З МІНІМАЛЬНИМ ЧАСОМ ПРОСТОЮ І МАКСИМАЛЬНОЮ ЕФЕКТИВНІСТЮ КОМАНД РОЗРОБКИ ..... 30
3.1. Вимоги до системи ..... 30
3.2. Опис системи ..... 31
3.3. Аналіз інструментів та засобів для розробки ..... 31
3.4. Архітектура розробленого програмного забезпечення ..... 37
3.5. Аналіз розробленої системи ..... 45
3.6. Висновки до розділу ..... 46
4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ ..... 47
4.1. Тестування програмного забезпечення ..... 47
4.2. Тестування алгоритму розподілення задач ..... 48
4.3. Оцінка результатів роботи розробленої системи ..... 49
4.4. Порівняння отриманих результатів з існуючими програмними аналогами ..... 58
4.5. Пропозиції для покращення системи ..... 59
4.6. Висновки до розділу ..... 60

5. ПОБУДОВА БІЗНЕС-МОДЕЛІ ..... 61
5.1. Опис проблеми ..... 61
5.2. Аналіз зацікавлених сторін ..... 62
5.3. Інноваційне програмне рішення ..... 63
5.4. Комерційний програмний продукт. Конкурентні переваги програмного продукту ..... 64
5.5. Ринок аналогічних програмних рішень ..... 65
5.6. Унікальна ціннісна пропозиція ..... 65
5.7. Потоки доходів. Структура витрат ..... 66
5.8. Канва бізнес-моделі стартапу ..... 67
5.9. Висновки до розділу ..... 69
ВИСНОВКИ ..... 70
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ ..... 72
ДОДАТКИ ..... 75

## СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

*Розподіл задач* – це процес визначення оптимального розподілу різних задач між групою виконавців з метою мінімізації загального часу виконання.

*Транспортна задача* — це специфічна задача лінійного програмування, застосовувана для визначення найбільш економічного плану перевезення однорідної продукції від постачальників до споживачів.

*Алгоритм апроксимації Фогеля* – це метод вирішення транспортної задачі, модифікована версія якого використовується для вирішення задачі розподілення задач.

*NP проблеми* – задачі, правильність рішення яких можна перевірити за поліноміальний час.

*Клас складності NP* — клас складності, до якого належать задачі, що можна розв'язати недетермінованими алгоритмами за поліноміальний час.

*Недетермінований алгоритм* – це алгоритм, який передбачає декілька шляхів обробки одних і тих самих вхідних даних, без будь-якого уточнення який саме варіант буде обраний.

*Евристичний підхід* – практичний метод вирішення проблем, який не гарантує ідеальних результатів.

*Граф* – структура даних, що представляє вузли та їх зв'язки.

*Оптимізація* – процес вдосконалення для досягнення найкращого можливого результату.

*Комбінаторика* – це галузь математики, що займається вивченням комбінацій об'єктів відповідно до певних правил.

*Апроксимація* – наближене визначення або оцінка значень.

*SDK (Software Development Kit)* – набір інструментів для розробки програмного забезпечення. Зазвичай представлений у вигляді окремих бібліотек, доступних для використання іншими розробниками.

*API (Application Programming Interface)* – інтерфейс для взаємодії між різними програмами.

# ВСТУП

У сучасному світі ефективне управління проектами та ресурсами стає ключовим фактором успіху для бізнесу. З огляду на це, розробка оптимізованого алгоритму для розподілу задач стає важливою задачею, яка вимагає інноваційного підходу. Ця магістерська дисертація пропонує новий алгоритм для ефективного розподілу задач у проектах, використовуючи поєднання евристичних методів і точних підходів.

Особливістю цього підходу є здатність обробляти великі обсяги даних та адаптуватися до різних сценаріїв виконання завдань.

Основними функціями, які підтримує запропонований алгоритм, є:

- аналіз і розподіл завдань з урахуванням їх часу виконання;
- балансування навантаження між виконавцями;
- адаптація до змін у проектах та ефективний перерозподіл ресурсів.

У дисертації проведено детальний аналіз існуючих підходів та методів, виявлено їх недоліки і на основі цього розроблено власний метод. Метою цієї роботи є розробка та валідація алгоритму, який зможе динамічно обирати спосіб оптимізації порядку виконання задач залежно від розміру набору даних.

Спосіб, запропонований у цій роботі, відрізняється своєю універсальністю та адаптивністю, роблячи його використання зручним як середніх та малих розмірів даних.

# 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ДЛЯ ПЛАНУВАННЯ ПОРЯДКУ ВИКОНАННЯ ЗАДАЧ ТА ЇХ РОЗПОДІЛЕННЯ

## 1.1. Аналіз основних методів планування порядку виконання задач

Планування порядку виконання завдань та їх розподілення є важливим аспектом у сфері планування роботи, яке має прямий вплив на ефективність робочого процесу та загальний результат проекту. Для оптимізації цього процесу в інформаційних технологіях наразі застосовуються людські ресурси та велика кількість спеціальних зустрічей робітників та менеджерів, де узгоджуються задачі та їх порядок виконання.

Такі задачі, через свою NP складність, стають схожими на задачу комівояжера, де необхідно знайти найкоротший шлях між пунктами. Пунктами у випадку завдання дисертації стають задачі, а вагами – час їх виконання.

Алгоритми перебору, генетичні алгоритми та інші евристичні підходи застосовуються для наближеного рішення цих складних оптимізаційних завдань. Сучасні підходи до планування завдань можна звести до графових моделей, де задачі формалізуються як вершини (або ноди), а залежності між ними як зв'язки. Така формалізація дозволяє використовувати вже існуючі алгоритми для пошуку оптимальних рішень.

У цьому розділі буде розглянуто існуючі рішення для планування порядку виконання завдань, їх особливості, функціонал, який пропонують дані системи для мінімізації часу простою, а також методи та алгоритми, які використовуються для роботи з графами та задачами оптимізації.

## 1.2. Огляд існуючих рішень

Цей розділ зосереджений на аналізі систем оптимізації. Даний аналіз є критично важливим, оскільки необхідно зрозуміти, як існуючі рішення використовують алгоритми для роботи з графами та алгоритми оптимізації для планування та виконання задач.

Ці системи оптимізації, як правило, застосовуються у широкому спектрі сфер, включаючи логістику, виробництво, транспортування та інші області, де необхідно ефективно розподіляти ресурси, планувати маршрути або вирішувати інші складні оптимізаційні завдання. Вони пропонують різноманітні підходи до рішення задач, що варіюються від лінійного та цілочисельного програмування до складніших алгоритмів, здатних обробляти великі обсяги даних та забезпечувати високий рівень точності в розподілі задач.

Система, розроблена в рамках магістерської роботи, фокусується на подібних аспектах оптимізації. Розглядаються, які дозволяють ефективно розподіляти задачі, забезпечуючи оптимальне використання ресурсів та мінімізацію загального часу виконання проекту. Це включає в себе аналіз та модифікацію існуючих алгоритмів, а також розробку нових підходів, що відповідають специфічним вимогам проекту

Наступним пунктом є список існуючих рішень від конкурентів.

### *1.2.1. OptaPlanner*

OptaPlanner – це високоефективний інструмент для рішення задач планування, який забезпечує широкі можливості для оптимізації часу виконання задач. Приклад рішення розподілення задач у цій системі зображено на рис. 1.

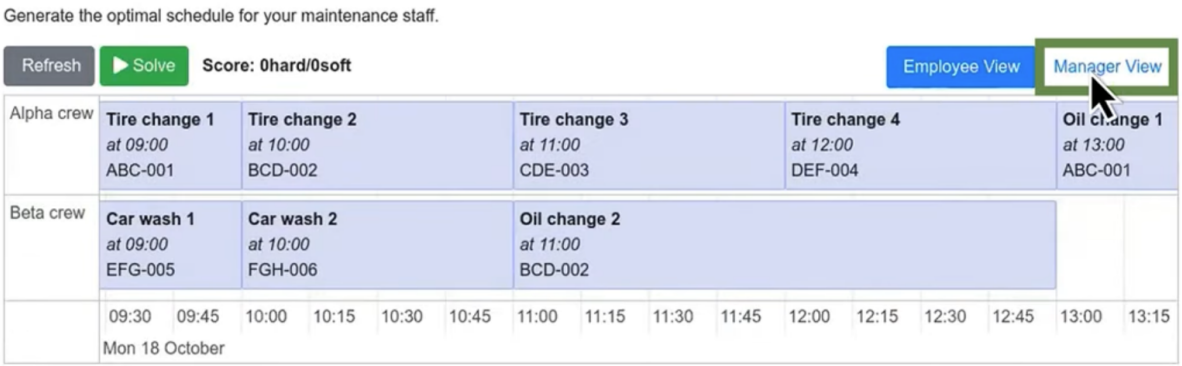



Рис. 1. Приклад розподілення задач в OptaPlanner

Він підходить для комплексних планувальних задач, таких як розклади, маршрутизація, або розподіл ресурсів. Цей інструмент використовує алгоритми, що базуються на правилах, що дозволяє ефективно знаходити оптимальні рішення для складних задач. Його гнучкість і масштабованість роблять OptaPlanner ідеальним вибором для бізнес-застосувань, що потребують динамічного планування.

OptaPlanner є ідеальним для широкого спектру планувальних задач, включаючи логістику та розклади, тоді як розроблена система зосереджена на максимізації ефективності розподілу задач. Він пропонує більш широкий набір функцій і гнучкість, але це може бути складніше у вивченні та налаштуванні. Розроблена система пропонує просту, але водночас ефективну альтернативу, зосереджену на конкретній задачі.

У той час як OptaPlanner вимагає більш глибокого розуміння алгоритмів оптимізації, розроблена система може бути більш привабливою для користувачів, що шукають більш прямолінійне рішення без необхідності заглиблення у складні аспекти планування.

У підсумку, розроблена система пропонує спеціалізоване рішення, яке може бути більш ефективним у певних сценаріях порівняно з більш універсальним, але складнішим OptaPlanner.

### *1.2.2. Google OR-Tools*

Google OR-Tools – це потужний набір інструментів для оптимізації комбінаторних завдань з відкритим вихідним кодом, який використовується у широкому спектрі додатків, від логістики до промислового планування. Цей інструмент пропонує різноманітні алгоритми для лінійного програмування, цілочисельного програмування, маршрутизації та інших оптимізаційних завдань. Google OR-Tools забезпечує високу продуктивність та можливість масштабування, що робить його відмінним інструментом для рішення складних оптимізаційних задач.

Google OR-Tools призначений для широкого спектру оптимізаційних завдань і підходить для різноманітних сценаріїв в логістиці, виробництві та інших областях, тоді як розроблена система спеціалізується на конкретному сегменті: оптимізації розподілу задач. Він надає більше можливостей у термінах технічної гнучкості та широти застосування. Натомість, розроблена система пропонує спеціалізоване рішення, що оптимізується для конкретної задачі.

OR-Tools може вимагати більш глибоких технічних знань для ефективного використання, в той час як розроблена система буде зосереджена на простоті та зручності для кінцевого користувача.

Загалом Google OR-Tools – це високоефективний інструмент для рішення широкого спектру оптимізаційних задач, а розроблена система надає більш цільове та просте рішення для специфічної задачі розподілу задач. Вибір між ними залежить від конкретних потреб проекту та вимог до функціональності.

### *1.2.3. CPLEX Optimization Studio*

CPLEX Optimization Studio від IBM – це інтегроване рішення для рішення задач лінійного та цілочисельного програмування.

Цей інструмент відомий своєю високою продуктивністю та здатністю ефективно розв'язувати великі та складні моделі оптимізації. CPLEX Optimization Studio часто використовується в промисловості та фінансах для оптимізації різноманітних процесів, від ланцюгів поставок до фінансового планування [15].

CPLEX Optimization Studio використовується для широкого спектру оптимізаційних задач і є ідеальним для складних проблем, де потрібно розглядати численні обмеження та змінні. Натомість, розроблена система спрямована на конкретну задачу – оптимізацію розподілу задач. Приклад вікна CPLEX Optimization Studio зображено на рис. 2.

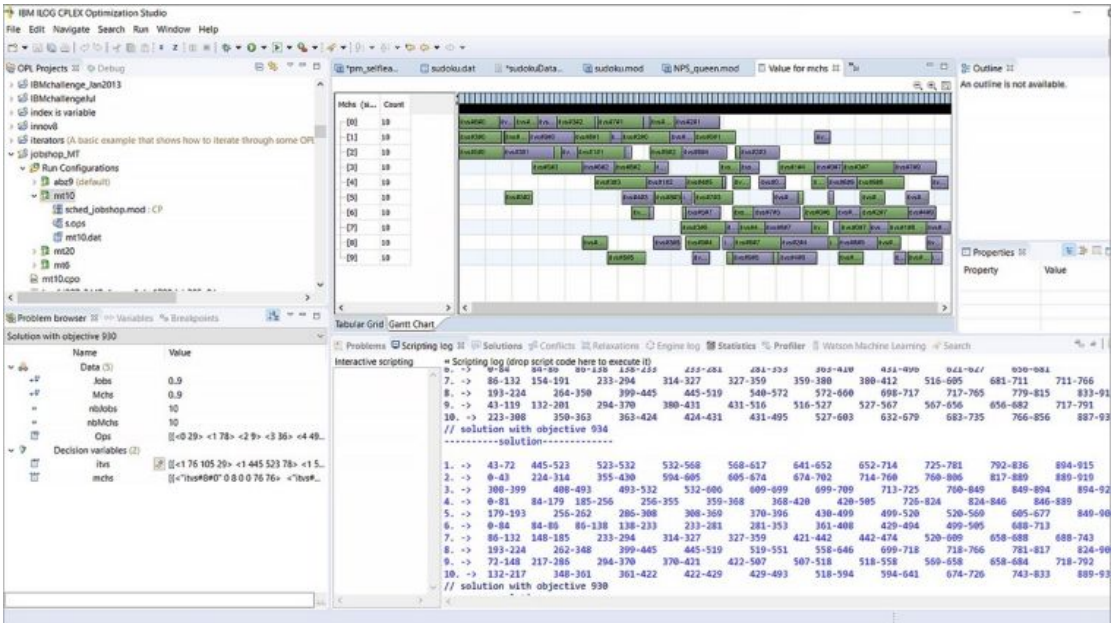


Рис. 2. Приклад вікна роботи CPLEX Optimization Studio

CPLEX Optimization Studio використовується для широкого спектру оптимізаційних задач і є ідеальним для складних проблем, де потрібно розглядати численні обмеження та змінні. Натомість, розроблена система спрямована на конкретну задачу – оптимізацію розподілу задач.

CPLEX пропонує розширені аналітичні та оптимізаційні можливості, вимагаючи від користувача глибоких знань у сфері оптимізації. Розроблена система ж зосереджена на більш вузькому аспекті, пропонуючи більш прямолінійне рішення.

Рішення від IBM надає гнучкість у інтеграції з іншими системами та мовами програмування, але вимагає технічної експертизи. Розроблена система забезпечує простоту використання та швидку адаптацію до потреб користувачів.

В цілому, CPLEX є потужним інструментом для широкого спектру оптимізаційних завдань, тоді як розроблена система зосереджена на специфічній задачі оптимізації розподілу задач, надаючи простішу, але не менш ефективну альтернативу. Рішення про вибір залежить від конкретних цілей і вимог проекту.

## 1.3. Основні метрики для аналізу

В рамках дослідницької роботи з оптимізації порядку виконання задач, виокремлюються основні метрики, за якими визначається вибір алгоритму:

- час виконання — оцінюється, який з алгоритмів забезпечує швидке виконання та здатність знайти оптимальний порядок виконання задач у короткі терміни.

Існуючі алгоритми, які обираються для вирішення задачі:

- алгоритми перебору;
- алгоритм мінімального кістякового дерева;
- генетичні алгоритми;
- алгоритми пошуку шляху;
- алгоритми для вирішення задачі транспортування.

### *1.3.1. Алгоритми перебору*

Алгоритми перебору – це підхід до розв'язання проблем, який полягає в систематичному перевірці всіх можливих комбінацій або варіантів для знаходження оптимального рішення. Вони часто використовуються в задачах оптимізації, де необхідно перебрати всі можливі варіанти для знаходження найкращого рішення.

В контексті задачі оптимального порядку виконання задач шляхом перебору всіх можливих варіантів, можна отримати повний набір можливих порядків виконання, і за допомогою певних критеріїв, таких як часові обмеження, мінімізація часу простою, тощо, визначити найкращий порядок. В результаті цього аналізу і використання алгоритмів перебору буде отримано можливість знайти оптимальний порядок виконання задач, що допоможе досягти найкращих результатів у виконанні проекту.

Алгоритми перебору є потужним інструментом для розв'язання оптимізаційних задач, проте вони можуть вимагати значних обчислювальних ресурсів і часу, особливо при великій кількості можливих

комбінацій, перестановок та розміщення. Часова складність перестановок зростає факторіально відносно розміру вхідних даних, оскільки для множини з $n$ елементів існує $n!$ можливих перестановок. Розміщення визначає кількість способів, якими можна розмістити $k$ елементів з множини розміру $n$, і має часову складність $O(\frac{n!}{(n-k)!})$. У випадку комбінацій, які репрезентують способи вибору $k$ елементів з множини розміру $n$ без урахування порядку, складність становить $O(\frac{n!}{k!(n-k)!})$. Тому, в цій роботі, будуть проводитися додаткові дослідження та оптимізацію алгоритмів перебору для забезпечення ефективності обчислень.

Переваги алгоритмів – простота реалізації, отримання всіх можливих результатів.

Недоліки – велика часова складність.

### *1.3.2. Алгоритм мінімального кістякового дерева*

Алгоритм мінімального кістякового дерева є алгоритмом графової оптимізації, який використовується для побудови підграфу, що містить всі вершини графу з найменшою сумарною вагою ребер [14]. Приклад його роботи можна побачити на рис. 3.

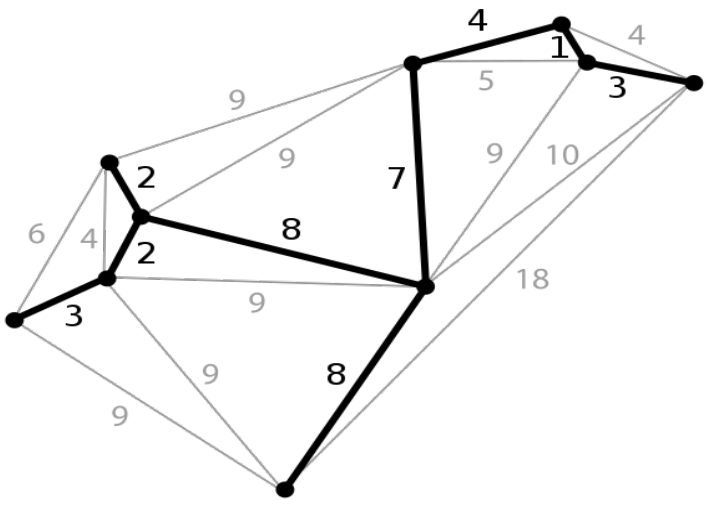



Рис. 3. Огляд результату роботи алгоритму мінімального кістякового дерева

Алгоритм мінімального кістякового дерева (МКД) використовується для знаходження мінімального кістякового дерева в зваженому зв'язному графі. Мінімальне кістякове дерево – це підграф даного графа, який включає всі вершини графа, не має циклів та має мінімальну суму ваг ребер.

Зв'язний граф – це граф, в якому між будь-якою парою вершин існує шлях $G(V, E)$, де $V$ – множина вершин, а $E$ – множина ребер. Для кожного із ребер $(u, v) \in E$ маємо його вагу $w(u, v)$. МКД для такого графа є підграф $T(V, \bar{E})$, де $\bar{E} \subseteq E$ і повна вага якого є найменшою $w(T) = \sum_{(u,v)\in T} w(u, v)$ .

Два найвідоміші алгоритми для знаходження МКД – це алгоритми Прима та Крускала [12]. Вони між собою відрізняються. Порядок роботи алгоритму Прима має такий вигляд:

1. Ініціалізувати дерево з однієї довільної вершини $v$.
2. Додавати до дерева найлегше ребро, що з'єднує вершину із дерева та вершину поза деревом.
3. Повторювати поки всі вершини не будуть включені до дерева.

Часова складність алгоритму Прима залежить від того, які структури даних використовуються для його реалізації. Для матриці суміжності це $O(V^2)$. Однак із використанням бінарних куп або куп Фібоначчі, складність може бути знижена до $O(E \log V)$ або $O(E + V \log V)$ відповідно.

Алгоритм Крускала має такий порядок дій:

1. Сортувати ребра графа за зростанням їх ваг: $e_1, e_2, \ldots, e_m$, де $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_m)$.
2. Ініціалізувати пусте дерево.
3. Для $i = 1$ до $m$. Якщо додавання ребра $e_i$ до $T$ не створює циклу в $T$, то додати $e_i$ до $T$.
4. Повернути $T$ як МКД.

Складність алгоритму з використання бінарної купи $O(E + V \log V)$.

Застосовуючи алгоритм мінімального кістякового дерева до задачі в магістерській роботі, можна визначити оптимальну послідовність

виконання задач. Дерево, яке утворюється за допомогою цього алгоритму, містить мінімальну кількість з'єднань та найкоротші шляхи між задачами. Таким чином, використання алгоритму мінімального кістякового дерева дозволить досягти оптимального порядку виконання задач у магістерській роботі, забезпечуючи ефективність та економію ресурсів.

Можна використати 2 алгоритми, оскільки вони повністю вирішують задачу та є достатньо швидкими.

Переваги алгоритмів:

- отримання оптимальної послідовності;
- ефективність та часова складність обох алгоритмів досить висока.

Недоліки алгоритмів:

- швидкість потребує нетривіальної реалізації через бінарну купу.

### *1.3.3. Генетичні алгоритми*

Генетичні алгоритми є еволюційними алгоритмами, які моделюють процес природного відбору для розв'язання оптимізаційних задач. Вони працюють за аналогією з генетичними процесами, такими як спарювання, схрещування та мутація, щоб створити нові покоління розв'язків [13].

Генетичні алгоритми використовують популяцію індивідів, які представляють потенційні розв'язки задачі. Ці індивіди піддаються еволюційним операціям, таким як схрещування (комбінування частин розв'язків) та мутація (випадкові зміни), щоб створити нові нащадки. Якість цих нащадків оцінюється за допомогою функції пристосованості, і найкращі розв'язки відбираються для наступного покоління.

Процес еволюції продовжується до досягнення заданої умови зупинки, такої як досягнення оптимального розв'язку або вичерпання встановленої кількості поколінь. Генетичні алгоритми широко застосовуються у задачах оптимізації, розкладанні задач, розв'язанні задач маршрутизації та багатьох інших областях, де потрібно знайти найкращий розв'язок з великим простором можливих варіантів.

Використання генетичних алгоритмів у магістерській роботі може допомогти знайти оптимальні рішення для складних задач планування, розподілу ресурсів або оптимізації процесу виконання задач [3]. Ці алгоритми використовують оператори кросовера, мутації та селекції, щоб ефективно перебирати та вдосконалювати рішення з кожним наступним поколінням.

Змодельовані розв'язки можуть бути представлені як бінарні рядки або вектори, які кодують потенційні рішення задачі: $Solution = \{s_1, s_2, \dots, s_n\}$, де $s_i$ відповідає значенню і-го параметра розв'язку. Популяція індивідів представляє розв'язки $Population = \{P_1, P_2, \dots, P_n\}$. Процес оцінки якості кожного розв'язку відбувається за допомогою функції пристосованості $\text{Fitness}(P_i)$. На основі цієї пристосованості відбувається вибір розв'язків для схрещування. Схрещування (кросовер) комбінує частини двох "батьківських" розв'язків для створення "дитячого" розв'язку: $Child = \text{Crossover}(Parent1, Parent2)$. Мутаційний процес вносить невеликі випадкові зміни в розв'язок $\text{Mutate}(Child)$.

Наприклад, при використанні генетичних алгоритмів для оптимізації процесу виконання проекту, можна представити рішення у вигляді генетичного коду, який відповідає параметрам проекту. Алгоритм буде генерувати та оцінювати популяцію рішень, застосовуючи оператори кросовера та мутації, щоб отримати нові варіанти. За допомогою процесу селекції, генетичні алгоритми визначать найкращі рішення, які можна застосувати для оптимізації процесу виконання проекту.

Переваги алгоритму:

- можливість адаптувати до величезної кількості нелінійних та недиференційовних задач;
- генетичні алгоритми здатні адаптуватися до змін у задачі, навіть якщо вони були неочікуваними або невідомими на початковому етапі;
- керування умовами зупинки та іншими параметрами процесу.

Недоліки алгоритмів;

- складність налаштування**,** для деяких задач може бути важко визначити підходящі генетичні оператори або функцію пристосованості;
- може потребувати більше часу ніж простіші в реалізації алгоритми.

### *1.3.4. Алгоритми пошуку шляху*

Алгоритми пошуку шляху є методами для визначення оптимального шляху або маршруту між двома або більше точками у графі або мережі. Ці алгоритми базуються на розгляді різних варіантів шляхів та оцінці їхньої ефективності.

Одним з найпоширеніших алгоритмів пошуку шляху є алгоритм Дейкстри, який знаходить найкоротший шлях між вершинами графу, використовуючи ваги ребер. Інший популярний алгоритм – це алгоритм A* (A-star), який комбінує інформацію про відстань та оцінку до цільової вершини для вибору найкращого шляху.

Ці алгоритми використовуються у багатьох областях, включаючи навігацію, транспортні системи, маршрутизацію даних у комп'ютерних мережах та робототехніку, де необхідно знайти найкращий шлях або оптимальний маршрут для досягнення певної цілі.

***Алгоритм A-star***

Алгоритм A* (A-star) є алгоритмом пошуку шляху та графічною технікою оптимізації, яка використовується для знаходження найкоротшого шляху від заданої початкової вершини до цільової вершини в графі. Цей алгоритм використовує концепцію евристики для пришвидшення процесу пошуку шляху.

Основні етапи алгоритму A* включають оцінку функції $f(n)$ для кожної вершини $n$. $f(n) = g(n) + h(n)$, де $g(n)$ вартість шляху від початкової точки вершини до вершини $n$, а $h(n)$ – евристична оцінка вартості шляху від вершини $n$ до цільової вершини.

Для повноти розуміння природі алгоритму A* необхідно згадати, що він по своїй суті є розширенням BFS (пошуку в дереві в ширину) з додаванням евристичної інформації для напрямного і більш ефективного пошуку шляху до цільової вершини. Для даної задачі важливим є як простота алгоритму так і його швидкодія. Надаючи перевагу швидкодії, взято A* як один і основних алгоритмів для розгляду.

Часова складність A* залежить від евристики. В найгіршому випадку в необмеженому просторі розширення кількості нод є експоненційним для найкоротшого шляху $d$: $O(b^d)$, де $b$ – середнє число нащадків для ноди.

Переваги алгоритму:

- ефективність – алгоритм A* ефективний для широкого спектра задач;
- оптимальність – за умови, що використовується допустима евристика, алгоритм гарантовано знаходить оптимальний шлях;
- всесторонність – A* можна застосовувати в різних доменах, включаючи планування задач, штучний інтелект у іграх, робототехніку та інші.

Недоліки алгоритму:

- використання пам'яті – алгоритм може вимагати значного обсягу пам'яті, особливо для складних графів;
- залежність від евристики – якість евристичної функції може значно вплинути на ефективність алгоритму;
- вартість виконання – для складних задач з великим простором станів, витрати на виконання можуть бути досить високими.

***Алгоритм Дейкстри***

Алгоритм Дейкстри – це алгоритм пошуку найкоротшого шляху в графі з невід'ємними вагами ребер. Він починає з визначення початкової вершини і обчислює найкоротші шляхи до всіх інших вершин графа.

Алгоритм Дейкстри працює шляхом перегляду сусідніх вершин та оновлення їх відстаней, якщо знайдений шлях до них є коротшим. Він

продовжує цей процес до тих пір, поки не будуть оброблені всі вершини графа або досягнутий бажаний кінцевий шлях.

Алгоритми пошуку шляху в магістерській роботі можуть бути використані для визначення найкоротшого шляху або оптимального маршруту між точками. Вони застосовуються для розв'язання задач маршрутизації, навігації або планування маршрутів.

### *1.3.5. Алгоритми для вирішення задачі транспортування*

Задача перевезення є частиною лінійного програмування і зосереджується на визначенні найкращого способу транспортування товарів з M відправних точок до N місць прибуття [1]. У цьому контексті, відомо, що витрати на перевезення товарів з i-того пункту відправлення до j-того пункту призначення є відомими.

Задача транспортування є досить схожою на задачу даного дослідження [2]. Цілком імовірним варіантом є певна модифікація алгоритмів вирішення цієї задачі для цього випадку.

Існує декілька алгоритмів для вирішення транспортної задачі:

1. Метод північно-західного кута. Цей метод розпочинає вирішення задачі транспортування з верхнього лівого кута матриці вартості і послідовно рухається до правого нижнього кута. Він простий у застосуванні, але не завжди гарантує оптимальний план перевезення.
2. Метод апроксимації Фогеля [4]. Цей метод використовує диференціювання для знаходження штрафів або витрат, асоційованих із кожним можливим варіантом перевезення.
3. Метод мінімальної ціни. Цей метод полягає в послідовному виборі клітинок матриці вартості з мінімальною ціною перевезення одиниці товару.

Кожен з методів, які перераховано вище, пропонують початковий розподіл, який не є гарантовано найкращим можливим. Для того, щоб

привести його до оптимального результату існує метод потенціалів. Задача розподілення задач пропонує певні обмеження у виконанні робіт виконавцями, тому метод потенціалів може бути недостатньо гнучким і у даному випадку буде розроблено спеціальний метод фінального балансування результатів.

## 1.4. Висновки до розділу

На основі аналізу конкурентів та різноманітних алгоритмів можна висунути висновки.

Під час розгляду конкурентів розділ був сфокусований на системах оптимізації, таких як OptaPlanner, Google OR-Tools та CPLEX Optimization Studio, які відіграють ключову роль у вирішенні складних оптимізаційних завдань у різних галузях. Ці системи надають потужні інструменти для розв'язання задач лінійного програмування, цілочисельного програмування, маршрутизації та інших оптимізаційних викликів.

Головною перевагою цих систем є їх здатність ефективно обробляти великі обсяги даних та складні обмеження, пропонуючи гнучкість та високий рівень точності у розв'язанні задач. Ці інструменти користуються великою популярністю у бізнес, мають велику клієнтську базу та широкі можливості інтеграції з різними системами.

У контексті магістерської роботи, основним завданням є розробка алгоритму для оптимального розподілу задач, з акцентом на мінімізацію часу виконання та ефективне використання ресурсів. Дана система вирізняється зосередженням на специфічній задачі оптимізації, пропонуючи простіше та більш цільове рішення у порівнянні з більш загальними інструментами оптимізації. Це відкриває шлях для потенційної інтеграції побудованого алгоритму в інші системи, дозволяючи користувачам використовувати цю розробку як доповнення до існуючих оптимізаційних інструментів.

Щодо технічної складової висновків, то генетичні алгоритми ефективно підходять для реалізації нетривіальних задач на великих наборах даних, але їх ефективність може зменшуватися при виконанні специфічних завдань з обмеженим обсягом інформації або іншими жорсткими обмеженнями.

Алгоритми знаходження мінімального кістякового дерева відмінно підходять для роботи на середніх та великих наборах даних, де необхідно враховувати велику кількість зв'язків між елементами. Пошукові алгоритми на рівні з алгоритмами мінімального кістякового дерева є схожими і працюють в тому самому напрямку тільки з урахуванням специфічних умов. В нагоді також стає евристика та додаткові можливості алгоритму Дейкстра по знаходження шляху до кожної ноди в графі.

Майбутній алгоритм буде комбінованим і його можна розділити на декілька частин:

1. Для невеликих наборів даних (до 100 задач) алгоритм перебору, з урахуванням певного порядку задач, виглядає досить привабливо завдяки його простоті та детермінованості.
2. Для середніх та великих наборів даних один із алгоритмів МКД може надати кращі результати завдяки його здатності до глибокого аналізу зв'язків між елементами. Також непоганим варіантом є один з алгоритмів задачі транспортування. З урахуванням їх потенційно неідеального результату вони можуть принести результати дуже і дуже швидко. Додавання певної оптимізації та обробки результату може принести досить конкурентні результати.
3. На дуже великих наборах даних можна розглядати генетичний алгоритм як експериментальний підхід, хоча його ефективність є під питанням у контексті поставленої задачі.

Усі представлені алгоритми мають бути модифікованими для роботи зі структурою даних задач, та комплексних взаємозв’язків між ними.

Цей розділ зосереджено розробці та валідації алгоритму, який зможе динамічно обирати підхід в залежності від розміру та характеристик набору даних. Такий алгоритм буде спрямований на забезпечення оптимальних результатів порядку виконання задач, представленими в графі для мінімізації часу на виконання проектної задачі, комбінуючи переваги вже відомих методів оптимізації. Дане рішення може бути модифікованих і використаним не тільки в домені проектного менеджменту, а і у задачах пріоритизації та оптимізації роботи операційних та пошукових систем, систем роботи з мережами, тощо.

# 2. МОДИФІКОВАНИЙ АЛГОРИТМ РОЗПОДІЛЕННЯ ЗАДАЧ

## 2.1. Аргументація модифікованого алгоритму

У розробці модифікованого алгоритму вирішення транспортної задачі ключовим є його поділ на три частини: повний перебір, апроксимація Фогеля та балансування результатів. Повний перебір застосовується для невеликих наборів даних (до 50 виконавців * кількість робіт, далі ОР – одиниці роботи) через свою факторіальну складність. На цьому етапі генеруються всі можливі комбінації розподілу задач, які потім сортуються за пріоритетом, визначаючи найкращий розподіл на основі мінімізації максимального часу виконання.

Алгоритм апроксимації Фогеля використовується для оптимізації розподілу задач у випадках більшого набору даних (50-8000 ОР), забезпечуючи більшу швидкість обчислень та відносну точність, порівняно з повним перебором. Балансування результатів дозволяє додатково оптимізувати розподіл задач, забезпечуючи більш рівномірний розподіл робочого навантаження серед виконавців, із мінімальним впливом на загальний час виконання.

Головне обмеження полягає в розмірі вхідних даних, оскільки збільшення кількості даних може значно збільшити час обчислень. Таким чином, алгоритм потребує додаткового аналізу та оптимізації для великих наборів даних.

## 2.2. Модифікація алгоритмів для роботи зі списками задач та їх зв’язками

Цей розділ буде направлений на більш детальний покроковий опис розробленого алгоритму, який, як було описано в попередньому розділі, складається з 3 частин: повний перебір, апроксимація Фогеля, балансування результатів апроксимації.

### *2.2.1. Повний перебір*

Повний перебір є початковим і найбільш прямолінійним алгоритмом вирішення NP проблем. Він є корисним виключно на малій кількості вхідних даних, оскільки має факторіальну складність, що практично унеможливлює його використання на більших розмірах даних. Блок схему алгоритму можна знайти на рис. 4.

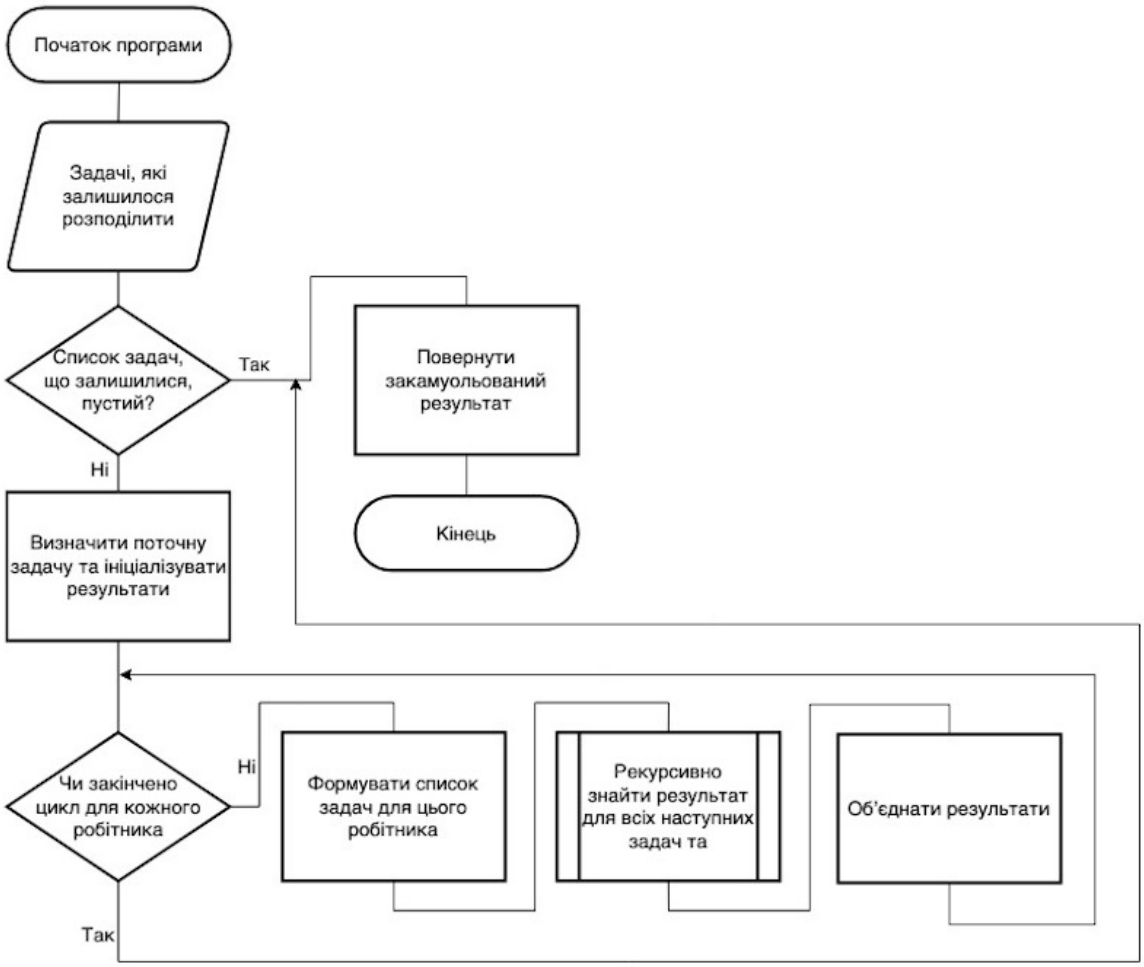



Рис. 4. Блок-схема алгоритму перебору

Етапи роботи алгоритму:

1. Генерація всіх можливих комбінацій розподілу задач. На цьому етапі алгоритм розглядає кожну можливу комбінацію розподілу задач між виконавцями з урахуванням списку виконавців, які можуть виконати певну задачу.

2. Сортування задач за пріоритетом. Після створення всіх комбінацій, задачі в кожній комбінації сортуються за пріоритетом. Цей етап не є обов'язковим, оскільки він впливає тільки на швидкість виконання і не змінює результат роботи алгоритму, оскільки сумарний час виконання не змінюється від зміни порядку доданків, а саме часу виконання робіт.
3. Вибір найкращої комбінації. З усіх можливих розподілів алгоритм вибирає той, що мінімізує максимальний час виконання задач серед усіх виконавців. Це досягається шляхом порівняння максимального часу виконання кожного можливого розподілу і вибору найкращого.

### *2.2.2. Алгоритм апроксимації Фогеля*

Алгоритм апроксимації Фогеля є одним із методів вирішення задачі транспортування і є кращим за інші такі як Алгоритм північно-західного кута, та Алгоритму Least Cost. Блок-схема алгоритму Фогеля зображена в додатку 1.

Кроки алгоритму Фогеля:

1. Обчислення штрафів. Для кожного рядка та стовпця визначаються дві найменші витрати, і розраховується їх абсолютна різниця. Різниця обчислюється за формулою: $Penalty_i = |c_{i1} - c_{i2}|$, де $c_{i1}$ та $c_{i2}$ – дві найменші витрати в рядку або стовпці.
2. Вибір осередку з максимальним штрафом. Визначається рядок або стовпець з максимальним штрафом $D$. У цьому рядку або стовпці обирається клітинка з найменшою вартістю для розподілу.
3. Коригування поставок та попиту. Якщо розподіл задовольняє попит $d_j$ або поставку $s_i$, відповідний рядок або стовпець вилучається з подальшого розгляду.
4. Ітерація та завершення. Процес повторюється, поки всі поставки та попити не будуть задоволені (тобто, коли вартість поставки в

кожному походженні та попит на кожному призначенні дорівнює нулю), після чого алгоритм завершується.

У випадку оновленого алгоритму певним чином будуть змінені вхідні дані. Кожен виконавець стає постачальником, тобто $s_j$, а кожна задача стає $d_j$, тобто буде отримувати від робітника певну кількість годин, яка їх потрібна. Кожен працівник матиме теоретично необмежену кількість годин, яку він може надати. Також для тих задач, які працівник виконати не може через обмеження з боку задач, потреба у часі буде максимальною, тобто умовно нескінченною, що зробить дану пару нецікавою для алгоритму. Таким чином отримано вироджену транспортну задачу, де для задоволення всього постачання часу, коли усі задачі отримають своїх виконавців, буде створене певне пусте місце постачання ($t_{dummy}$) для симетризації задачі. З точки зору таблиці постачання і споживання це матиме наступний вигляд:

Таблиця 1

Приклад звичайної таблиці для транспортної задачі

| | $d_1$ | $d_2$ | $d_3$ | $d_4$ | Постачання |
|---|---|---|---|---|---|
| $s_1$ | 3 | 2 | 7 | 6 | 50 |
| $s_2$ | 7 | 5 | 2 | 3 | 60 |
| $s_3$ | 2 | 5 | 4 | 5 | 25 |
| Потреба | 60 | 40 | 20 | 15 | |

Таблиця 2

Приклад модифікованої таблиці для транспортної задачі

| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_{dummy}$ | Постачання часу |
|---|---|---|---|---|---|---|

Продовження табл. 2

| | | | | | | |
|---|---|---|---|---|---|---|
| $w_1$ | 2 | 3 | INF | INF | INF | INF |
| $w_2$ | INF | INF | 4 | 5 | INF | INF |
| $w_3$ | INF | 3 | 4 | INF | INF | INF |
| Потреба часу | 2 | 3 | 4 | 5 | INF | |

Часова складність алгоритму апроксимації Фогеля $O(k(nm\ log(mn))$, де $k$ – кількість ітерацій головного циклу, а $m$ і $n$ кількість рядків і стовпців. В найгіршому випадку $k$ буде мати таке саме значення як $m$n, оскільки доведеться обробляти кожен елемент таблиці. Для даної задачі було введено поняття ОР – одиниці роботи, тому, використовуючи символ $W = nm$, отримано таке значення найгіршої швидкодії $O(W^2log(W))$, що являє собою квазівадратичну складність у найгіршому випадку.

## 2.3. Особливості модифікації алгоритму

Результат роботи алгоритму транспортування балансує між оптимальністю і швидкістю роботи, що є припустимим для такого типу задач. Для вирішення проблеми недостатнього балансування запропоновано додатковий алгоритм фінального балансування результатів.

Балансування результатів апроксимації є останнім етапом евристичного алгоритму. Він викликається тільки для результатів алгоритму Фогеля, оскільки для повного перебору балансування не потрібне. Етапи роботи алгоритму балансування:

1. Визначення кількості виконавців для балансування. Стандартно для балансування беруться всі виконавці.

2. Розрахунок часу для кожного виконавця. Для кожного виконавця обчислюється загальний час виконання призначених йому задач.
3. Сортування виконавців за часом виконання задач. Виконавці сортуються від тих, хто має найменший час виконання, до тих, у кого час найбільший.
4. Перерозподіл задач між виконавцями. Алгоритм проходиться по виконавцям з найбільшим часом виконання та намагається перерозподілити задачі між ними та виконавцями з найменшим часом, з метою зменшення різниці в загальному часі роботи.
5. Оновлення часу виконання задач. Після кожного перерозподілу задач час виконання для виконавців оновлюється.

Цей алгоритм дозволяє більш рівномірно розподілити навантаження між виконавцями, що може призвести до оптимізації загального часу виконання всіх задач. Часова складність алгоритму $O(W\log_2(W) + W_mT)$, де $W$ – кількість виконавців, $W_m$– кількість виконавців, які балансуються, $T$ – сумарна кількість задач. Просторова складність O(1), оскільки усі зміни відбуваються з уже готовими наборами даних без створення нових великих об’єктів.

Забігаючи наперед на 4 розділ про аналіз отриманих результатів, потрібно зазначити, що балансування показує себе дуже непогано на невеликих наборах даних, які є допустимими для розробленого алгоритму і показує зміну часу виконання алгоритму на +-2-5% від результату без балансування, що в свою чергу можна навіть списати на похибку тестового середовища (персональний комп’ютер з декількома іншими програмами запущеними паралельно, які можуть нерівномірно завантажувати систему в різний момент часу).

Ще однією особливістю є розділення алгоритму на певні складові в залежності від розміру вхідних даних. Це досить стандартна методика оптимізації алгоритмів і дозволяє досягти вагомих результатів при правильному розподілі і використанні усіх доступних ресурсів системи. Під

використанням усіх доступних ресурсів мається на увазі потенційне збільшення граничного порогу роботи повного перебору при збільшенні обчислювальних потужностей або розпаралелення роботи алгоритму на декілька потоків.

Блок-схема результуючого алгоритму роботи програми зображена на рис. 5.

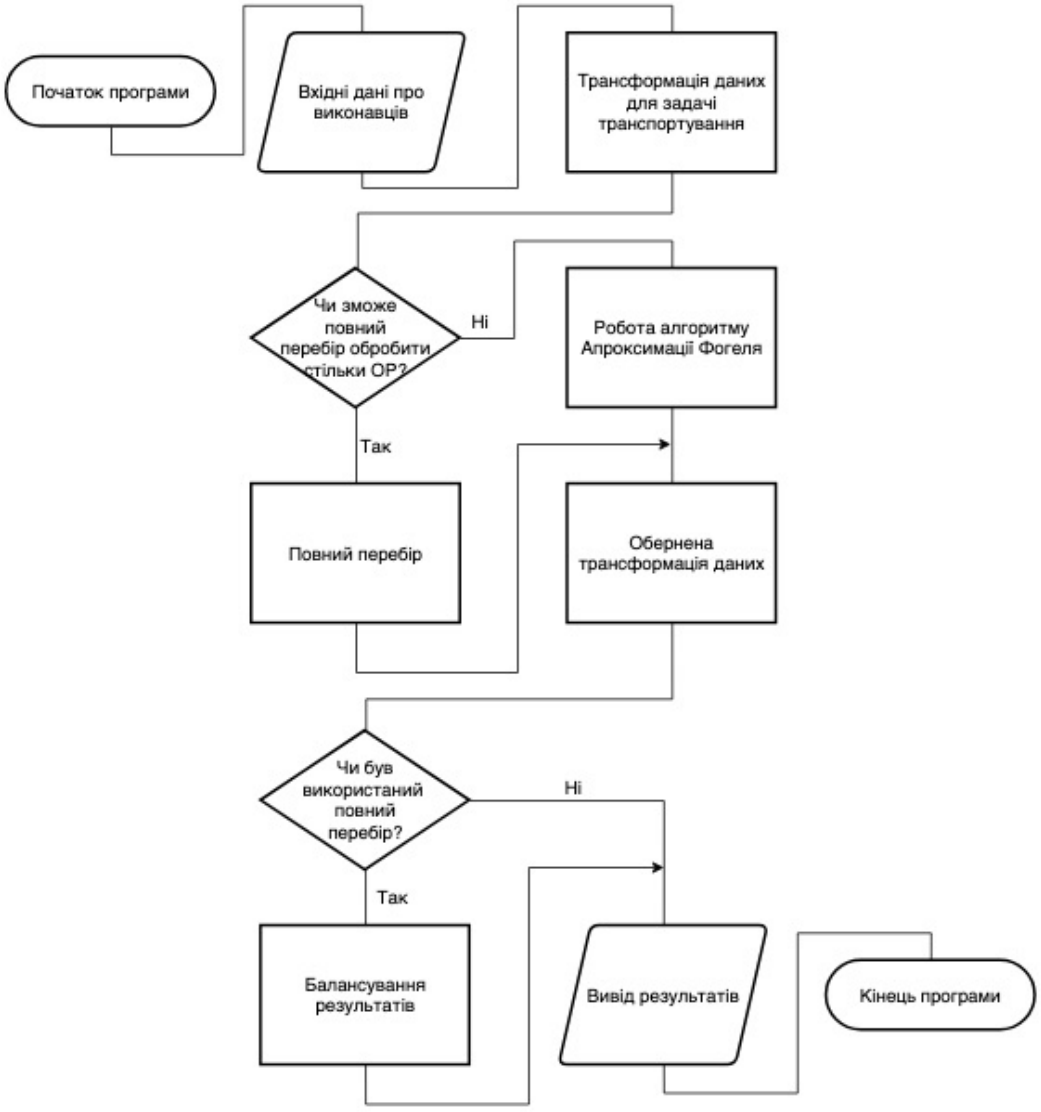



Рис. 5. Блок схема алгоритму роботи програми

## 2.4. Висновки до розділу

У даному розділі було розроблено алгоритм для розподілення задач між виконавцями. Він є модифікацією існуючого алгоритму для розв’язання

транспортної задачі: алгоритму Фогеля. Було також розділено алгоритм на 3 складові, які працюють в комбінації в залежності від розміру вхідних даних. Дана методика дозволить реагувати на кількість елементів, які треба обробляти та дозволяє модифікувати та покращувати кожен окремий алгоритм, який використовується. Прикладом такої модифікації є алгоритм балансування результатів роботи алгоритму Фогеля, який досить непогано швидкодію та приносить користь там, де попередній алгоритм ставить у пріоритет швидкість перед точністю результатів [5].

# 3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ АЛГОРИТМУ РОЗПОДІЛЕННЯ ЗАДАЧ В ЧАСІ З МІНІМАЛЬНИМ ЧАСОМ ПРОСТОЮ І МАКСИМАЛЬНОЮ ЕФЕКТИВНІСТЮ КОМАНД РОЗРОБКИ

У даному розділі надано детальний опис програмних компонентів, з яких складається система.

## 3.1. Вимоги до системи

Функціональні вимоги:

1. Реалізація алгоритму розподілу задач. Основою системи є алгоритм, який дозволяє оптимально розподіляти задачі між працівниками, виходячи з їхньої доступності та вимог задач. Алгоритм забезпечує ефективне управління часом та ресурсами, мінімізуючи загальний час виконання проекту.
2. Консольний додаток. Початкова версія системи реалізована як консольний додаток. Це дозволяє зосередитися на алгоритмічній складовій та забезпечує можливість подальшого розвитку системи до повноцінної програми з фронтендом і бекендом.
3. Вхідні дані з JSON файлу. Система приймає вхідні дані у форматі JSON, що спрощує інтеграцію з різними джерелами даних та забезпечує гнучкість у використанні.
4. Виведення результатів у JSON файл. Результати роботи алгоритму виводяться у файл JSON, забезпечуючи легке інтегрування з іншими системами та можливість подальшої обробки даних.

Нефункціональні вимоги:

1. Модульна архітектура – система розроблена з урахуванням модульної архітектури, що дозволяє з легкістю модифікувати та розширювати її функціональність. Ця властивість є критично важливою для адаптації системи до змінюваних вимог та умов роботи.

Ці вимоги відображають стратегічний підхід до розвитку системи, де основний акцент робиться на розв'язанні ключових алгоритмічних задач, з залишенням простору для подальшої еволюції системи.

## 3.2. Опис системи

Розроблювана система представляє собою консольний застосунок, спрямований на ефективний розподіл задач серед команди працівників. Цей підхід вибрано для забезпечення максимальної гнучкості та зосередження на алгоритмічному рішенні, яке є ключовим аспектом проекту. Користувач має можливість ввести дані про задачі та працівників через консоль, після чого система виконує обчислення та генерує оптимальний план розподілу робіт.

Завдяки модульній архітектурі, розроблений консольний застосунок має потенціал для подальшого розширення та інтеграції з іншими системами. У майбутньому це може включати розвиток веб-інтерфейсу, що робить систему більш доступною та зручною для широкого кола користувачів

## 3.3. Аналіз інструментів та засобів для розробки

Було обрано реалізацію проекта у вигляді консольного застосунку, який теоретично можна розширювати в сторону окремого виділеного бекенду.

### *3.3.1. Javascript*

JavaScript, часто вживана веб-розробниками мова програмування, є інтерпретованою та заснованою на концепції об'єктно-орієнтованого програмування. Ця мова характеризується гнучкістю та динамічною типізацією, що робить її популярною серед розробників для створення інтерактивних веб-сайтів та додатків. Основними характеристиками JavaScript є:

- гнучкість та динамічна типізація – JavaScript дозволяє використовувати змінні без необхідності заздалегідь задекларувати їх типи, що полегшує швидку розробку;
- прототипне успадкування – відмінною особливістю JavaScript є його система прототипного успадкування, яка надає гнучкість у роботі з об'єктами;
- асинхронність – JavaScript має потужні можливості для асинхронного програмування, особливо з використанням промісів та асинхронних функцій (async/await);
- широка підтримка – Javascript має величезну кількість пакетів відкритого коду для розширення, що забезпечує підтримку спільноти, постійне розширення та доповнення функціоналу та дружелюбність мови для новачків.

Перевагами Javascript є:

- універсальність – JavaScript можна використовувати як на клієнтській, так і на серверній стороні (Node.js), що дозволяє створювати повністю інтегровані веб-додатки;
- багата екосистема – наявність величезної кількості бібліотек та фреймворків (React, Angular, Vue.js) робить JavaScript однією з найбільш функціональних мов для веб-розробки;
- легкість інтеграції – JavaScript легко інтегрується з іншими технологіями та мовами програмування, зокрема HTML та CSS;
- велика спільнота – однією з ключових переваг є велика та активна спільнота розробників, яка постійно розвиває мову та підтримує новачків.

JavaScript підтримує модульну структуру, дозволяючи розробникам організовувати код у вигляді відокремлених модулів, що полегшує управління великими проектами та сприяє повторному використанню коду. Мова також підтримує розширюваність, дозволяючи включати сторонні бібліотеки для розширення функціональності.

JavaScript є невід'ємним для створення сучасних застосунків. Він використовується для додавання інтерактивності, створення анімацій, обробки даних форм, валідації на стороні клієнта та багато іншого. Завдяки Node.js, JavaScript також став популярним вибором для серверного програмування [6].

JavaScript є гнучкою, могутньою та широко використовуваною мовою програмування, що робить її ідеальним вибором для розробки як клієнтських, так і серверних частин веб-додатків. Його універсальність, інтеграція з іншими технологіями, розширені можливості та велика спільнота роблять його незамінним інструментом для будь-якого сучасного веб розробника.

### *3.3.2. Typescript*

TypeScript є сучасною мовою програмування, розробленою та підтримуваною компанією Microsoft, яка служить надбудовою до JavaScript. TypeScript додає статичну типізацію до JavaScript, що є однією з ключових особливостей, розширюючи можливості та підвищуючи продуктивність розробки великих застосунків [9].

Основні характеристики TypeScript:

- статична типізація – TypeScript дозволяє розробникам задавати типи змінних, параметрів функцій, значень, які повертаються, тощо, що забезпечує більшу безпеку коду, допомагаючи уникнути типових помилок, які можуть виникати в JavaScript;
- сумісність з JavaScript – оскільки TypeScript є надбудовою до JavaScript, весь синтаксис та код JavaScript є повністю сумісним з TypeScript, що робить перехід на TypeScript відносно безболісним для існуючих проектів JavaScript;
- інструменти розробки – TypeScript покращує розробку завдяки підтримці інтелектуальної автодоповнення коду, рефакторингу та більш точним повідомленням про помилки в процесі компіляції;

- підтримка ООП – TypeScript підтримує об'єктно-орієнтоване програмування з класами, інтерфейсами, успадкуванням тощо, надаючи розробникам потужні засоби для побудови структурованого коду.

Переваги використання TypeScript:

- підвищення якості коду – статична типізація допомагає виявити помилки на ранніх етапах розробки, спрощуючи процес відлагодження та підтримки коду;
- праща підтримка великих проектів – TypeScript ідеально підходить для розробки великих та складних застосунків, де потрібен високий рівень організації коду та управління типами;
- велика спільнота та екосистема – TypeScript швидко набув популярності серед розробників, завдяки чому має велику спільноту та широкий спектр ресурсів для навчання та підтримки;
- інтеграція з сучасними фреймворками – TypeScript відмінно інтегрується з популярними фреймворками та бібліотеками, такими як Angular, React, та Vue.js.

TypeScript стає все більш вибором для розробки застосунків, де потрібна більша надійність та структурованість коду. Він особливо корисний в розробці великих застосунків, де важливо забезпечити легкість підтримки та масштабування.

TypeScript, з його сумісністю з JavaScript, статичною типізацією, підтримкою об'єктно-орієнтованого програмування, та інтеграцією з сучасними фреймворками, становить собою потужний інструмент для розробки високоефективних веб-застосунків. Використання TypeScript у вашому проекті розподілення задач в часі з мінімальним часом простою та максимальною ефективністю команд розробки є ідеальним вибором, який забезпечує якість, надійність та легкість управління кодом.

### *3.3.3. NodeJS*

Node.js, значуще середовище виконання JavaScript на сервері, відіграє ключову роль у сучасній веб-розробці. Його використання в проекті з розподілу задач є стратегічним рішенням, що обумовлене його простотою, широкою підтримкою та ефективністю при роботі з алгоритмами [6].

Основні характеристики Node.js:

- однопоточність з подієвою моделлю – Node.js використовує асинхронну, неблокуючу модель вводу-виводу, що робить його особливо ефективним для обробки великої кількості паралельних з'єднань без значного навантаження;
- використання JavaScript – Node.js дозволяє розробникам використовувати JavaScript для написання серверного коду, забезпечуючи єдиність мови програмування для всього стеку веб-додатків;
- велика екосистема – завдяки npm (Node Package Manager), Node.js має одну з найбільших екосистем бібліотек та модулів, що надає розробникам величезний набір інструментів.

Переваги використання Node.js у проекті:

- легкість розробки – простота та інтуїтивність Node.js, разом з великою кількістю доступних ресурсів і документації, робить його ідеальним для швидкої розробки та експериментування з алгоритмами;
- продуктивність – Node.js підходить для оптимізації алгоритмів розподілу задач, забезпечуючи високу продуктивність завдяки ефективній обробці асинхронних подій.

В майбутньому, зі зростанням вимог до продуктивності та складності проекту, може виникнути необхідність переходу на більш продуктивні та низькорівневі мови програмування, такі як Rust, C/C++, Go, Dart або Java. Це дозволить оптимізувати виконання складних алгоритмів та підвищити загальну продуктивність системи.

Node.js виявився ефективним вибором для реалізації алгоритму розподілення задач, забезпечуючи простоту розробки та достатню продуктивність для аналізу алгоритмічної складової. Його гнучкість та широке застосування в сучасній веб-розробці роблять його ідеальним кандидатом для використання у даному проекті.

### *3.3.4. Faker.JS*

Faker.js є важливим інструментом у світі сучасної веб-розробки, надаючи розробникам легкий спосіб генерування масивних об'ємів реалістичних випадкових даних. Завдяки його здатності імітувати різноманітні сценарії використання, від імен та адрес до випадкових текстових рядків, Faker.js стає неоціненним для тестування та прототипування [8]. Його гнучкість і простота використання дозволяють в значній мірі спростити процес розробки, давши можливість зосередитися на більш важливих аспектах проекту, не витрачаючи час на ручне створення тестових даних.

Загалом, Faker.js є незамінним інструментом у арсеналі веб-розробника, що значно полегшує процес розробки та забезпечує високу якість кінцевого продукту. Його використання дозволяє ефективно моделювати різні випадки використання програми, забезпечуючи реалістичність та різноманітність даних, необхідних для всебічного тестування.

### *3.3.5. Jest*

Jest, розроблений Facebook, став одним з найпопулярніших фреймворків для тестування у світі JavaScript. Його простота використання, швидкість і потужні можливості роблять Jest ідеальним вибором для тестування як невеликих, так і великих масштабних JavaScript проектів. Завдяки своїй конфігурації "з коробки", Jest забезпечує негайне тестування без необхідності складного налаштування, що робить його доступним навіть для початківців у розробці [7]. Його можливості мокування (mocking) та

шпигунства (spying) дозволяють детально контролювати як виконання тестів, так і поведінку залежностей.

Однією з ключових особливостей Jest є його ізольоване середовище тестування, яке забезпечує надійність тестів та запобігає побічним впливам між тестами. Це особливо важливо при роботі з великими проектами, де кожен компонент або модуль має бути протестований окремо. Крім того, Jest включає у себе потужні інструменти для визначення покриття коду, що є важливим аспектом для забезпечення якості коду, дозволяючи розробникам визначити, які частини коду не охоплені тестами.

Jest також має велику спільноту та широку підтримку, завдяки чому він постійно оновлюється і покращується. Його інтеграція з іншими популярними фреймворками та бібліотеками, такими як React, Vue.js та Angular, робить його ще більш привабливим для сучасних веб-розробників. Враховуючи ці переваги, Jest є відмінним вибором для будь-якого проекту, де важливо швидке та ефективне тестування**.**

### 3.4. Архітектура розробленого програмного забезпечення

Дане програмне забезпечення було побудоване у вигляді консольного застосунку на базі NodeJS. Основним завданням було імплементація алгоритму та простота взаємодії з середовищем роботи. Основними етапами роботи програми є:

1. Генерація даних.
2. Робота алгоритму.
3. Виведення результатів роботи.

На рис. 6 можна побачити порядок основного процесу виконання застосунку.

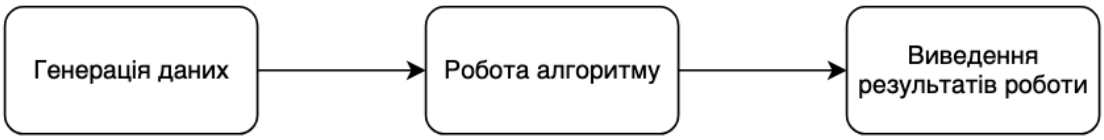



Рис. 6. Діаграма процесу виконання застосунку

Генерація даних є окремим процесом, який можна виконувати незалежно від самої роботи алгоритму і навпаки – алгоритм може бути запущеним на уже згенерованих даних. Для опису кожного з етапів початково представлено структури даних.

### *3.4.1. Моделі даних*

У даному програмному рішенні було описано певні основні типи даних у Typescript для спрощення розробки. Нижче наведено основні з них. Починаючи з Worker описаному у лістингу 1.

Лістинг 1. Тип даних Worker

```
export type Worker = {
  id: string;
  name: string;
};
```

Тип даних Worker представляє собою виконавця або робітника у системі. Кожен виконавець характеризується унікальним ідентифікатором id та іменем name. Ідентифікатор забезпечує можливість однозначно визначати кожного виконавця в системі, а ім'я дозволяє легко ідентифікувати та відображати виконавців для користувачів системи.

Наступною основною моделлю даних є Task, яка описана у лістингу 2.

Лістинг 2. Тип даних Task

```
export type Task = {
  id: string;
  name: string;
```

```
  duration: number;
  canBeDoneBy: string[];
  priority: number;
};
```

Тип Task відображає задачу, яка має бути виконана. Кожна задача містить унікальний ідентифікатор id, назву name, тривалість duration, список ідентифікаторів виконавців, які можуть виконати задачу canBeDoneBy, та пріоритет priority. Тривалість задачі визначається як відносне число, яке може представляти години, дні або іншу одиницю часу. Пріоритет визначає важливість задачі у контексті інших задач.

Наступними основними типами даних є Input та Output описані у лістингу 3.

Лістинг 3. Тип даних Input

```
export type Input = {
  workers: Worker[];
  tasks: Task[];
};

export type Output = {
  result: Record<Worker['id'], Task[]>;
};
```

Тип Input є основою для вхідних даних алгоритму розподілу задач. Він містить два основні поля: workers та tasks. Поле workers є масивом об'єктів типу Worker, що представляє всіх виконавців, доступних для розподілу задач. Поле tasks є масивом об'єктів типу Task, що представляє всі задачі, які необхідно розподілити між виконавцями.

Тип Output визначає формат результату алгоритму розподілу задач. Він містить одне поле result, яке є записом (об'єктом), де ключами є ідентифікатори виконавців (Worker['id']), а значеннями – масиви задач (Task[]), що призначені кожному виконавцю. Ця структура дозволяє ефективно представляти розподіл задач з мінімальним можливим загальним часом виконання.

Структурно на вихід з роботи алгоритму отримано таку організацію даних зображену на рис. 7.

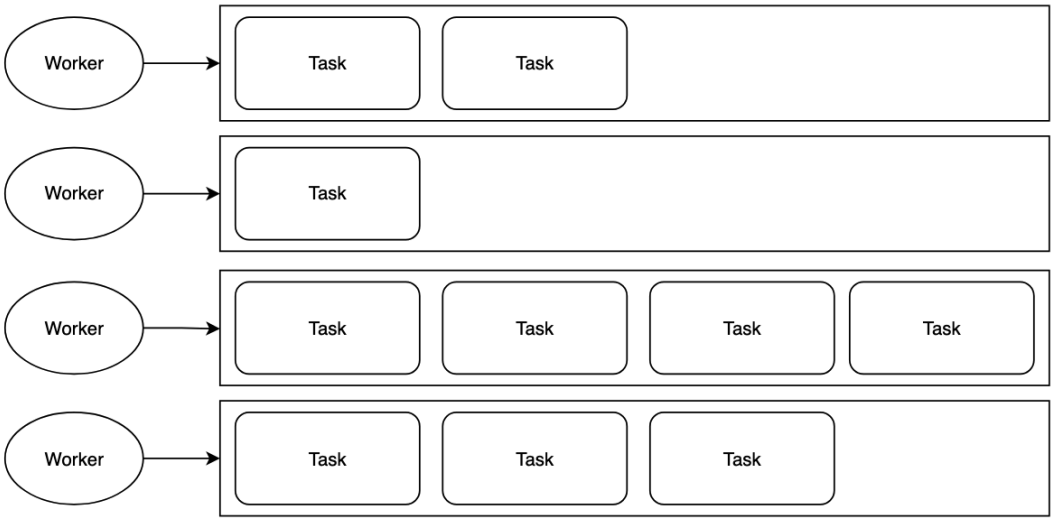



Рис. 7. Діаграма організації даних на виході роботи алгоритму

### *3.4.2. Внутрішня структура додатку*

Внутрішня структура додатку складається з різних модулів та методів, які взаємодіють один між одним. Одним із принципів розробки було зменшення зв’язності та збільшення пов’язаності модулів (більш сталі англомовні терміни – *coupling* та *cohesion*).

Зв'язність відноситься до ступеня, в якому один компонент або модуль програмного забезпечення залежить від інших модулів. Це означає, наскільки сильно модулі пов'язані між собою через обмін інформацією або використання функціоналу інших модулів. Зв'язність важлива для розуміння взаємодії між різними частинами програми та їх залежності один від одного. Більш наочно термін зв’язності можна на рис. 8.

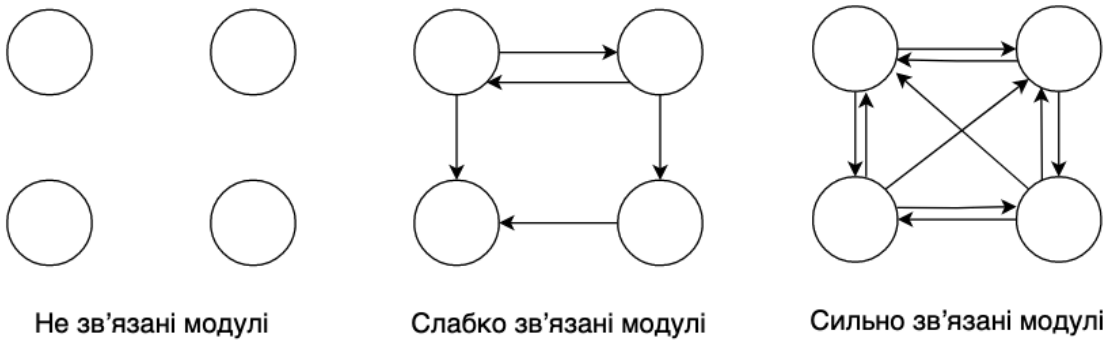



Рис. 8. Пояснювальна діаграма зв’язності модулів

Термін "пов'язаність" відображає рівень, на якому елементи всередині одного модуля програми взаємопов'язані, наприклад, через спільну мету або функціональність. Наочний приклад пояснення пов’язаності у поєднанні зі зв’язністю зображений на рис. 9.

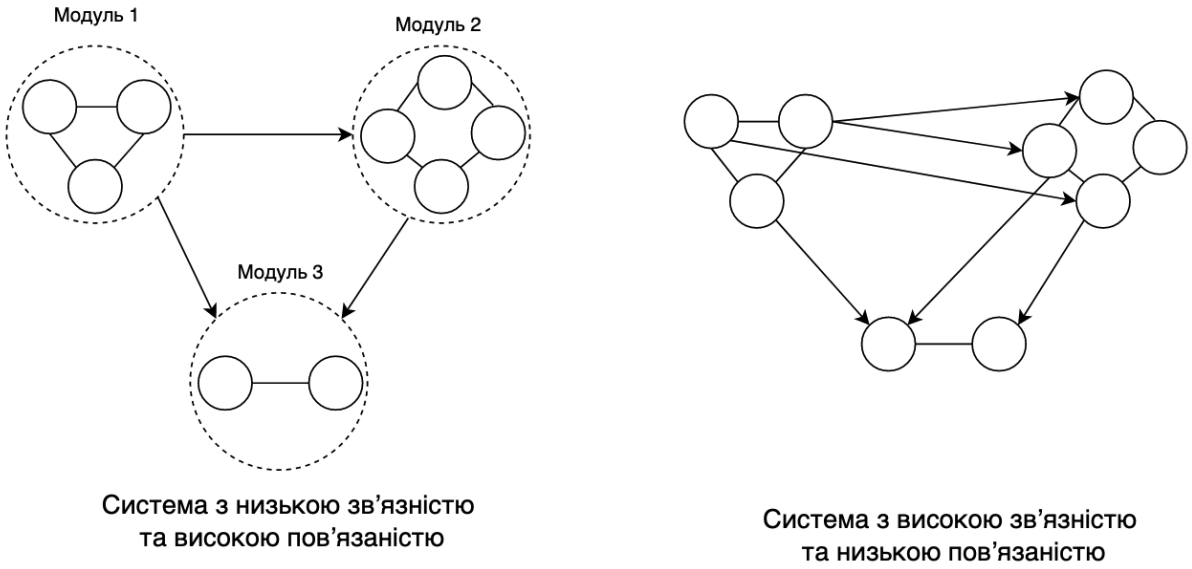



Рис. 9. Діаграма приклад пояснення пов’язаності у поєднанні зі зв’язністю

Рухаючись до програмних модулів, з яких складається система, нижче описано перший модуль: модуль генерації даних.

***Модуль генерації даних***

Модуль генерації даних складається з декількох методів: generateTask, generateWorker і generateInput. Основним методом, який

використовується з цього модуля є generateInput, що є прикладом сильної пов’язаності елементів між собою (генерація тільки в одному модулі, який використовує generateTask і generateWorker) та слабкої зв’язності модулів, оскільки модуль генерації використовується тільки для створення псевдосправжніх даних. Його внутрішня реалізація може бути легко замінена на іншу бібліотеку або джерело даних і при цьому сам модуль так само буде слугувати для однієї цілі і матиме той самий інтерфейс взаємодії з іншими модулями системи. Огляд модуля можна побачити на рис. 10.

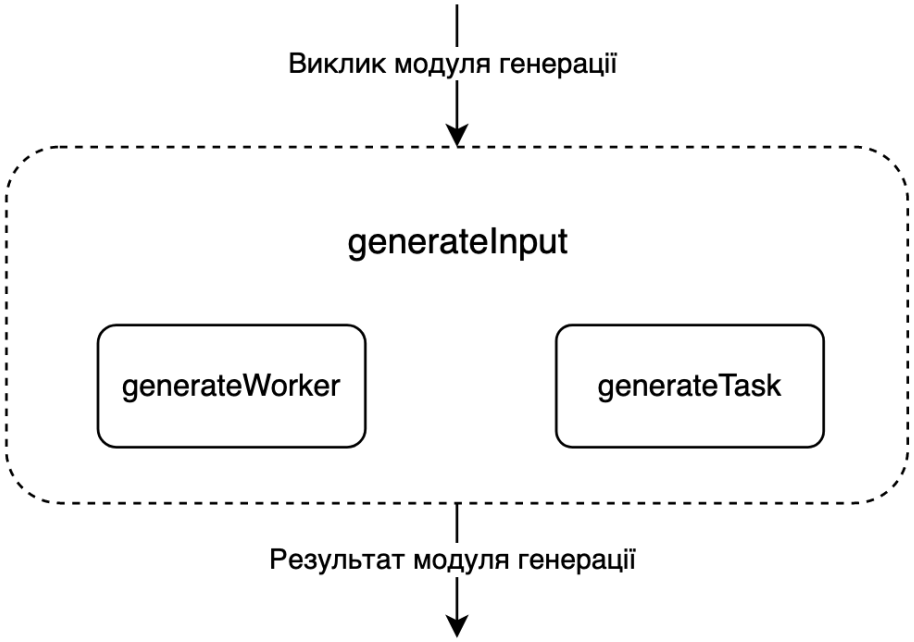



Рис. 10. Діаграма модуля генерації даних

***Модуль роботи алгоритму***

Модуль роботи алгоритму є основним програмним модулем, який всередині взаємодіє з декількома іншими меншими модулями. Цей модуль не лише служить як основний виконавчий компонент, але й координує ряд допоміжних модулів. Центральна роль цього модуля полягає у виконанні кількох ключових функцій.

На початковому етапі роботи алгоритму відбувається процес трансформації даних, необхідний для задачі транспортування. Цей етап

передбачає трансформацію вхідних даних у формат, придатний для обробки алгоритмом. Методи, які використовуються:

- *mapTasksInputToTransportationProblemInput* – трансформація моделі задачі розподілення завдань в модель для задачі транспортування;
- *mapTransportationProblemOutputToTasksOutput* – трансформація моделі задачі транспортування в модель задачі розподілення завдань.

Наступним етапом є робота самого алгоритму. Цей етап є серцем модуля, де відбувається власне виконання алгоритму розподілу задач. Алгоритм аналізує задачі та їх вимоги до виконавців, розподіляє задачі з урахуванням встановлених параметрів та оптимізує загальний час виконання. Методи, які використовуються:

- *vogelsApproximation* – алгоритм роботи задачі Фогеля;
- *brutForce* – алгоритм повного перебору.

Балансування результатів за необхідності. Після основної роботи алгоритму, може знадобитися балансування результатів для досягнення оптимального розподілу. Це включає перерозподіл задач між виконавцями з метою мінімізації часу простою та забезпечення рівномірного навантаження. Методи, які використовуються:

- *balance* – алгоритм балансування.

Ключовим аспектом архітектури цього модуля є його взаємозамінність етапів роботи. Кожен з етапів розроблено таким чином, щоб бути незалежним один від одного, що значно знижує зв'язність усієї системи. Така архітектура також сприяє легшому тестуванню та налагодженню, оскільки можливо ізолювати та перевіряти кожен етап окремо.

Ця структура графічно описана на рис. 11.

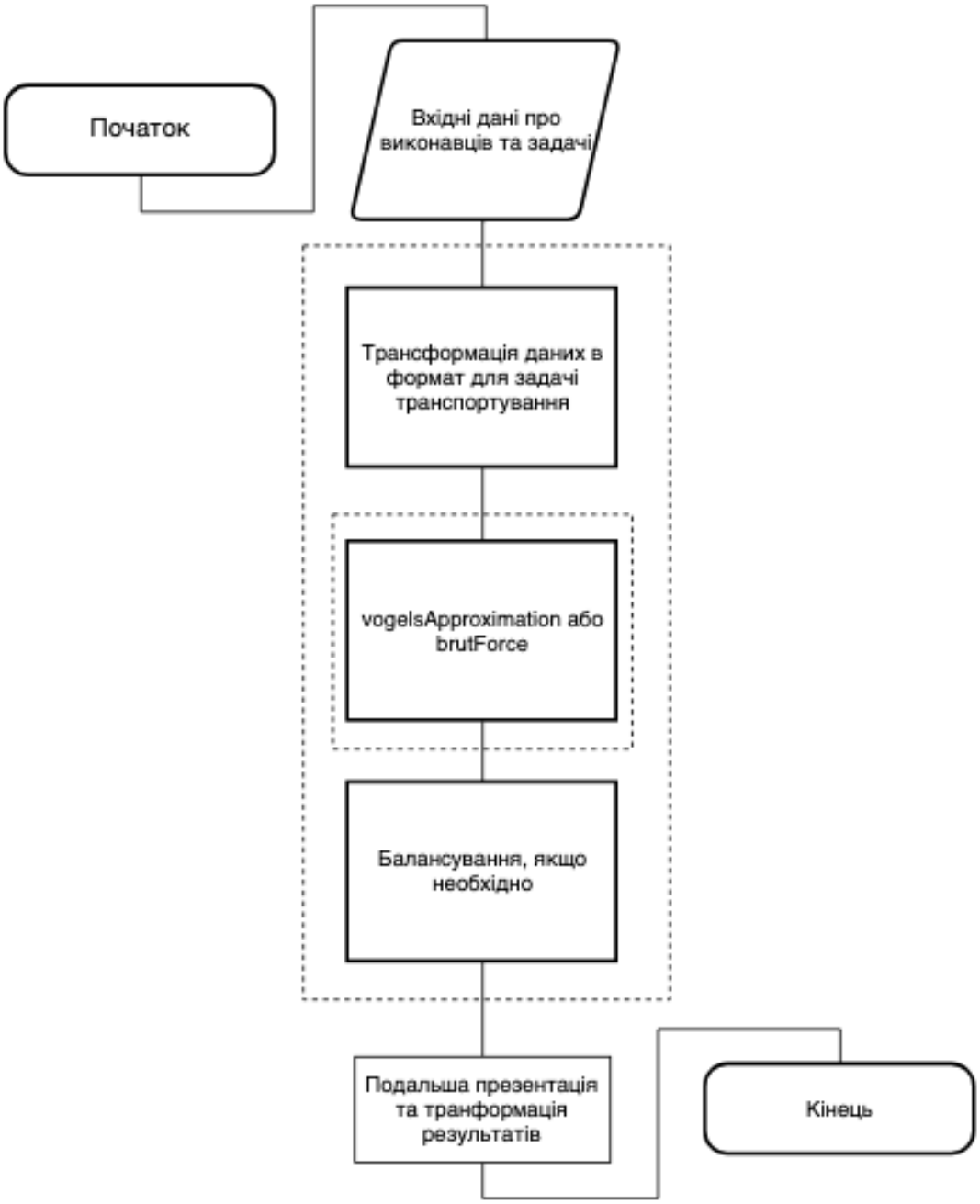



Рис. 11. Графічна репрезентація основного модуля роботи програми

***Модуль презентації результатів***

Модуль презентації результатів у розробленому програмному забезпеченні відіграє ключову роль у візуалізації та інтерпретації даних, отриманих від алгоритму розподілу задач. На поточному етапі розробки цей модуль реалізований у формі простого консольного виводу, що дозволяє швидко та ефективно переглянути результати виконання алгоритму. Такий

підхід є оптимальним для перевірки функціональності алгоритму та проведення первинних тестів. На рис. 12 можна побачити приклад графічного відображення результатів.

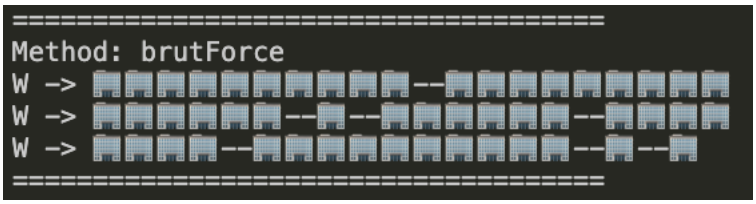



Рис. 12. Приклад графічної репрезентації результатів в консолі

Проте, в планах розвитку проекту передбачено значне розширення функціоналу модуля презентації результатів. Одним із напрямків є реалізація інтерактивної візуалізації за допомогою графів у браузері. Це дозволить не лише зробити результати більш наочними та зрозумілими для користувачів, але й надасть можливість детального аналізу розподілу задач, візуалізації навантаження на виконавців, а також ефективного відображення динаміки виконання задач в часі.

Інтеграція з сучасними веб-технологіями та використання інтерактивних елементів значно підвищить цінність розробленого програмного забезпечення, роблячи його не лише потужним інструментом для аналізу та планування, але й привабливим у використанні.

### 3.5. Аналіз розробленої системи

Розроблене програмне забезпечення представляє собою інструмент для оптимізації та ефективного розподілу задач між виконавцями. Цей інструмент складається з кількох ключових модулів, кожен з яких відіграє важливу роль у загальній архітектурі системи та її функціональності. Основні аспекти системи включають:

1. Модуль трансформації даних. Цей модуль відповідає за підготовку та трансформацію вхідних даних для подальшого аналізу. Він

забезпечує визначення відповідних атрибутів для задач і виконавців, що є критично важливим для точного розподілу задач.

2. Основний алгоритмічний модуль – ядро системи, де відбувається власне виконання алгоритму розподілу. Алгоритм обробляє задачі та розподіляє їх з урахуванням різних параметрів, забезпечуючи оптимізацію часу виконання, та балансує результати за необхідністю.
3. Модуль презентації результатів. На даному етапі розробки, цей модуль забезпечує просте відображення результатів у консолі. У майбутньому планується розширення функціоналу для створення більш наочних інтерактивних візуалізацій.

## 3.6. Висновки до розділу

У ході дослідження та розробки системи для оптимізації та розподілу задач було виконано аналіз потреб та сформульовано конкретні вимоги до додатку. Для реалізації проекту обрано мову програмування JavaScript у поєднанні з Node.js, що забезпечує гнучкість та широкі можливості для масштабування.

Розроблену систему характеризує використання комбінації алгоритмів для ефективного розподілу задач, включаючи алгоритм повного перебору для невеликих наборів даних та модифікований алгоритм апроксимації Фогеля для більших обсягів. Також важливу роль відіграє алгоритм балансування для оптимального розподілу завдань між виконавцями.

Особлива увага приділена розробці архітектури додатку, яка включає модулі трансформації даних, власне алгоритм, балансування, та презентацію результатів. Кожен модуль системи розроблений з урахуванням потреби в подальшому покращенні, розширенні функціональності та масштабуванні, що робить додаток гнучким та адаптивним до змінних вимог.

# 4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Даний розділ ознайомлює зі способом тестування та перевірки працездатності системи на різних наборах даних. Були знайдені певні обмеження, проведені дослідження різних варіантів роботи системи та порівняння з існуючими програмними аналогами.

## 4.1. Тестування програмного забезпечення

У процесі розробки та реалізації даного програмного засобу є певні особливості. Дана система була розроблена за допомогою розробки, яка керується тестами (далі TDD – від англ. Test Driven Development).

Test Driven Development (TDD) є методологією розробки програмного забезпечення, яка наголошує на важливості написання тестів до реалізації функціональності. Цей підхід не тільки сприяє створенню надійного та добре протестованого коду, але й допомагає утримувати систему гнучкою та легкою для модифікацій. На рис. 13 зображено структуру розробки при використанні TDD.

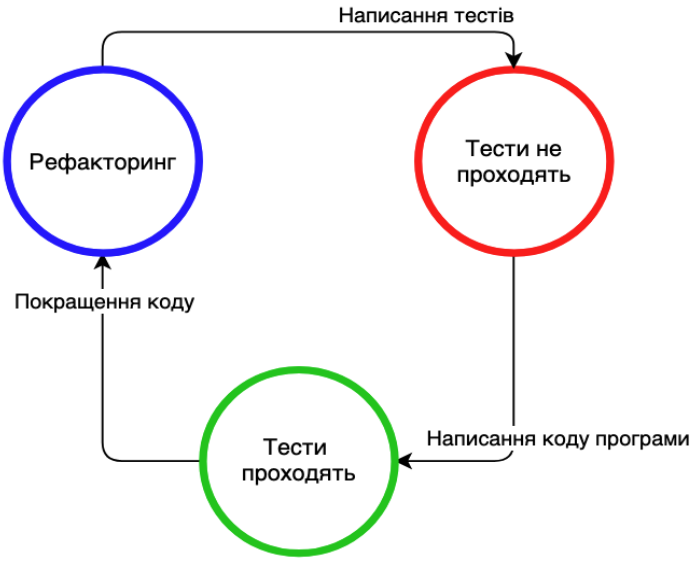



Рис. 13. Структура роботи під час розробки за методологією TDD.

Test Driven Development (TDD) є вельми вдалим підходом для виконання цієї магістерської роботи, зокрема через фокус на алгоритмах та розрахунках. Використання TDD в контексті роботи з чітко структурованими модулями та методами відповідає основним принципам цієї методології. Особливо ефективним TDD є в сценаріях, де потрібно проводити тестування маленьких, незалежних блоків коду, так званого unit тестування. Це дозволяє ретельно перевіряти кожну частину алгоритму на правильність, забезпечуючи високу надійність та якість загальної системи.

## 4.2. Тестування алгоритму розподілення задач

У даній роботі алгоритм розподілення задач був протестований за допомогою unit тестування. На малій кількості ОР, де використовується алгоритм повного перебору, результати роботи алгоритму порівнювалися з ручними розрахунками, що виступали як референсні значення. Цей підхід забезпечив високу точність тестування і сприяв швидкому виявленню помилок на ранніх етапах розробки. При тестуванні алгоритму Фогеля та алгоритму балансування, які активізуються при обробці більших наборів даних, було застосовано схожу стратегію. Для цього було обрано декілька репрезентативних наборів даних, для яких результати були розраховані заздалегідь. Приклад коду тесту зображений у лістингу 5.

Лістинг 5. Приклад unit тесту

```
it('should assign all tasks to a single worker if there is only one
worker', () => {
 const input: Input = {
   workers: [{ id: '1', name: 'Worker 1' }],
   tasks: [
     {
       id: '1',
       name: 'Task 1',
       duration: 1,
       canBeDoneBy: ['1'],
       priority: 1,
     },
     {
       id: '2',
       name: 'Task 2',
       duration: 2,
       canBeDoneBy: ['1'],
```

```
      priority: 1,
    },
  ],
};
const expectedOutput: Output = {
  result: {
    '1': [
      {
        id: '1',
        name: 'Task 1',
        duration: 1,
        canBeDoneBy: ['1'],
        priority: 1,
      },
      {
        id: '2',
        name: 'Task 2',
        duration: 2,
        canBeDoneBy: ['1'],
        priority: 1,
      },
    ],
  },
};
const actualOutput = method(input);
expect(actualOutput).toEqual(expectedOutput);
});
```

## 4.3. Оцінка результатів роботи розробленої системи

Швидкодію алгоритму виміряно у мілісекундах в залежності від розміру вхідних даних. Одиницею розміру вхідних даних є OP (одиниці роботи) як зазначено у попередніх розділах, що є добутком кількості робітників і кількості задач.

### *4.3.1. Генерація тестових даних*

Дані для дослідження швидкодії алгоритму були згенеровані за допомогою модуля генерації даних. Приклад згенерованих даних зображений у лістингу 6.

Лістинг 6. Приклад згенерованих даних

```
{
   "workers": [
     {
       "id": "f9121f60-c754-4d6f-8c14-14fbe4b5b2cb",
       "name": "pecus-sum-clamo"
     },
     {
       "id": "1dc3b85f-4e56-4449-82f8-02cbf811edf4",
       "name": "argumentum-vulariter"
     },
```

```
      {
        "id": "76dd5109-d01d-4c4e-8145-9ca03513c4a9",
        "name": "volo-vado"
      }
    ],
    "tasks": [
      {
        "canBeDoneBy": [
          "76dd5109-d01d-4c4e-8145-9ca03513c4a9"
        ],
        "duration": 8,
        "id": "36b96c89-b9d5-4e94-a93f-953281007188",
        "name": "Illum.",
        "priority": 2
      },
      {
        "canBeDoneBy": [
          "76dd5109-d01d-4c4e-8145-9ca03513c4a9",
          "1dc3b85f-4e56-4449-82f8-02cbf811edf4",
          "f9121f60-c754-4d6f-8c14-14fbe4b5b2cb"
        ],
        "duration": 5,
        "id": "f372b50b-aa55-461f-a3df-f826c3abe450",
        "name": "Impedit canto.",
        "priority": 10
      },
      {
        "canBeDoneBy": [
          "1dc3b85f-4e56-4449-82f8-02cbf811edf4",
          "76dd5109-d01d-4c4e-8145-9ca03513c4a9"
        ],
        "duration": 5,
        "id": "38ae0f5b-ab27-4c4c-84d2-ebd2eae2df22",
        "name": "Textilis adhuc carus cohaero villa.",
        "priority": 3
      },
      {
        "canBeDoneBy": [
          "1dc3b85f-4e56-4449-82f8-02cbf811edf4"
        ],
        "duration": 3,
        "id": "0728c4f1-2235-44a7-bf6e-ae26bc600fee",
        "name": "Totus.",
        "priority": 6
      },
      {
        "canBeDoneBy": [
          "1dc3b85f-4e56-4449-82f8-02cbf811edf4"
        ],
        "duration": 7,
        "id": "f5756d5f-b1c6-4f83-b9e9-8f372443eb53",
        "name": "Vos demergo adopto.",
        "priority": 10
      }
    ]
}
```


Тестові дані генеруються різних розмірів. Після емпіричного підбору кількості ОР, які здатна обробити розроблена система було обрано діапазон

0-8000. У разі виконання програми на більшій кількості елементів завершення за певний короткий обмежений час не може бути гарантовано.

### *4.3.2. Розрахунок часу виконання алгоритму*

Результати часу роботи алгоритму повного перебору в таблиці 3.

Таблиця 3

Результати роботи алгоритму повного перебору

| Кількість задач | Кількість працівників | OP | Час виконання (мс) |
|---|---|---|---|
| 5 | 3 | 15 | 0,13 |
| 10 | 3 | 30 | 0,76 |
| 10 | 5 | 50 | 12,9 |

Графічно переглянути час виконання алгоритму в залежності від одиниць роботи можна на рис. 14.

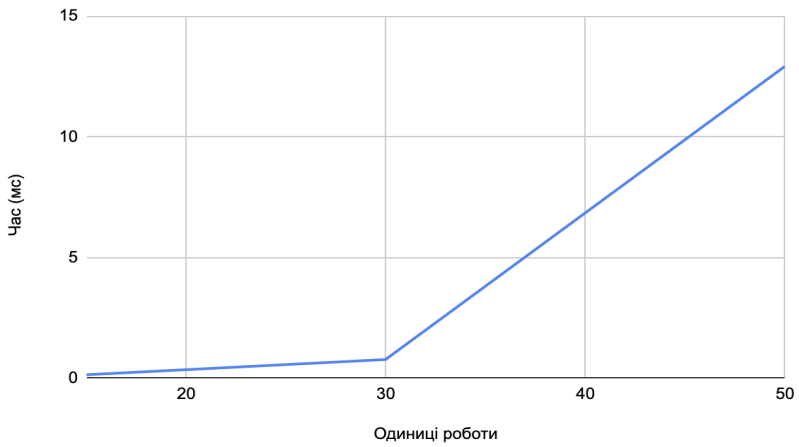



Рис. 14. Графічне зображення часу виконання алгоритму повного перебору на невеликих наборах даних

Одразу видно, як швидко зростає час виконання, враховуючи, що часова складність алгоритму є факторіальною. На більших розмірах даних при тестуванні на середньому комп'ютері система видавала помилку недостатньої кількості пам'яті.

Результати роботи алгоритму апроксимації Фогеля без балансування зображено в таблиці 4.

Таблиця 4

Результати часу роботи алгоритму апроксимації Фогеля без балансування

| Кількість задач | Кількість працівників | ОР | Час виконання (мс) |
|---|---|---|---|
| 10 | 3 | 30 | 0,26 |
| 10 | 5 | 50 | 0,1 |
| 20 | 10 | 200 | 0,52 |
| 30 | 20 | 600 | 2,6 |
| 100 | 30 | 3000 | 32,4 |
| 150 | 50 | 7500 | 116,9 |
| 160 | 50 | 8000 | 140,8 |

Графічно переглянути час виконання алгоритму в залежності від одиниць роботи можна на рис. 15.

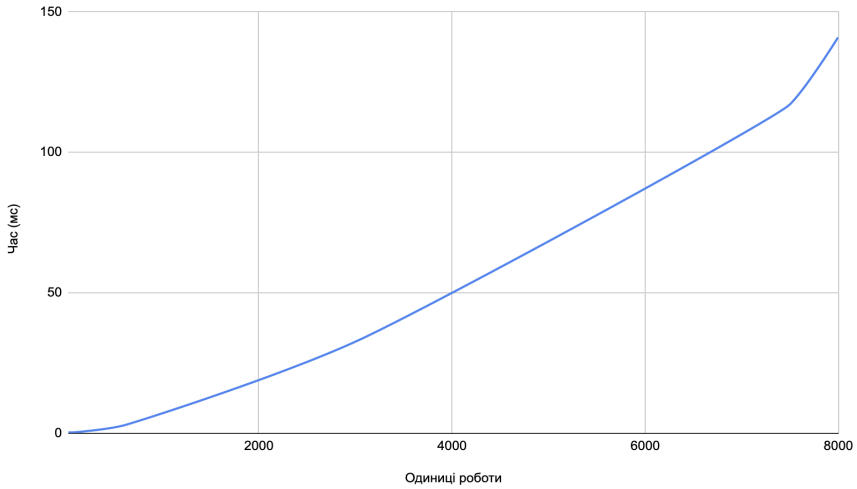



Рис. 15. Графічне зображення часу виконання алгоритму апроксимації Фогеля без балансування

В результаті дослідження швидкодії можна стверджувати, що при більш швидкому середовищі запуску алгоритм може себе показувати достатньо швидко і його складність з огляду на графік десь на рівні степеневої функції.

При додаванні балансування як додаткового етапі після алгоритму Фогеля отримано наступні результати описані в таблиці 5.

Таблиця 5

Результати виконання алгоритму Фогеля з балансуванням результатів

| Кількість задач | Кількість працівників | ОР | Час виконання (мс) |
|---|---|---|---|
| 10 | 3 | 30 | 0,41 |
| 10 | 5 | 50 | 0,23 |
| 20 | 10 | 200 | 0,5 |
| 30 | 20 | 600 | 4,02 |
| 100 | 30 | 3000 | 33,8 |

Продовження табл. 5

| | | | |
|---|---|---|---|
| 150 | 50 | 7500 | 125,32 |
| 160 | 50 | 8000 | 142,07 |

Графічно це зображено на рис. 16.

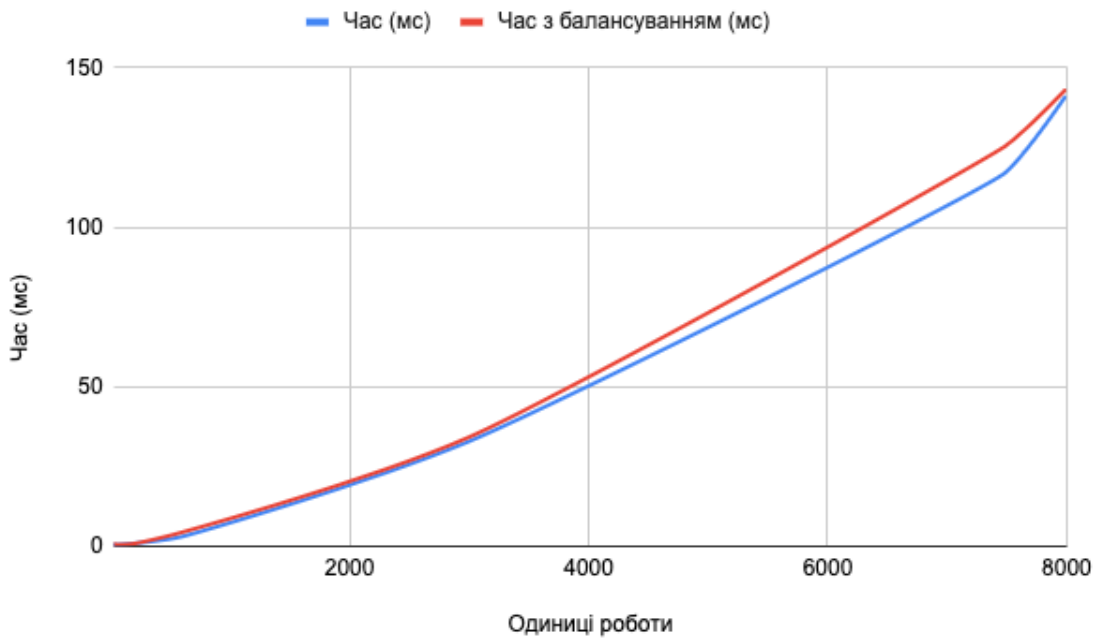



Рис. 16. Графічне зображення порівняння часу виконання алгоритму Фогеля з балансуванням після нього і без

З огляду на результати можна сказати, що балансування майже не змінює час виконання всього алгоритму, що є гарним значенням при покращенні результату роботи алгоритму. Зміна, яка відображена на графіку може навіть бути прийнята за похибку.

Дивлячись на прогнозовану найгіршу часову складність $O(W^2 log(W))$, де $W$ – це одиниці роботи, які було введено раніше, видно, що у графіка є явна тенденція на зростання у вигляді досить пологої параболи, що підтверджує прогнозовані розрахунки.

### *4.3.3. Аналіз точності роботи алгоритму*

Для аналізу ефективності роботи алгоритму було взято фактичні приклади результату роботи. Їх було візуалізовано в консолі. W – працівник і від нього уже список задач, які він буде виконувати. Довжина кожної задачі відповідає її часу виконання. Результат роботи алгоритму повного перебору зображений на рис. 17.

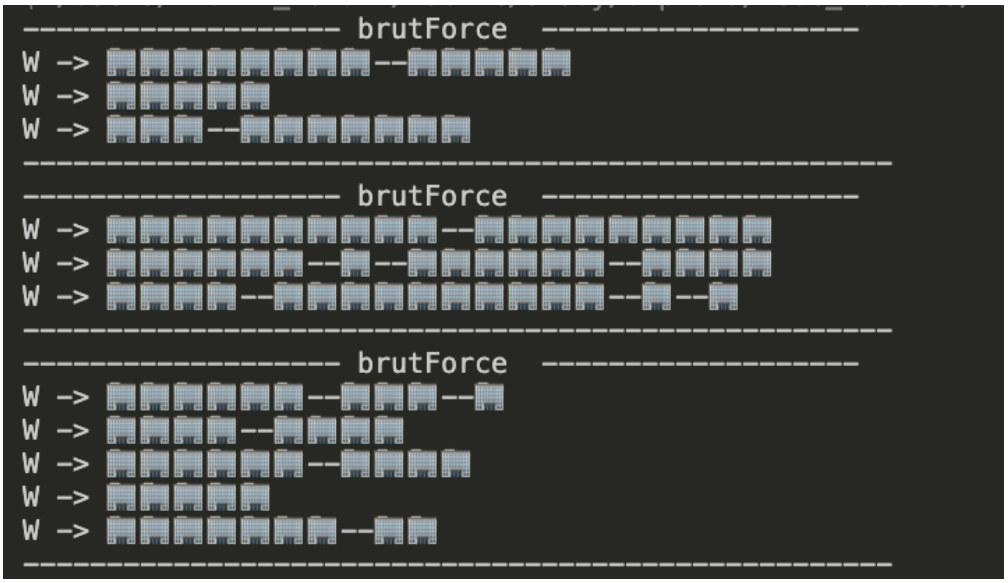



Рис. 17. Приклад розподілення задач між робітниками при повному переборі

Із зображення можна побачити, що більшість задач розподілені досить рівномірно і алгоритм справився зі своєю роботою. Враховуючи, що деякі задачі можуть бути виконані тільки певними робітниками, робимо висновок, що результат роботи є найбільш точним і не потребує покращення.

Переходимо до алгоритму апроксимації Фогеля. Результат роботи без балансування зображений на рис. 18.

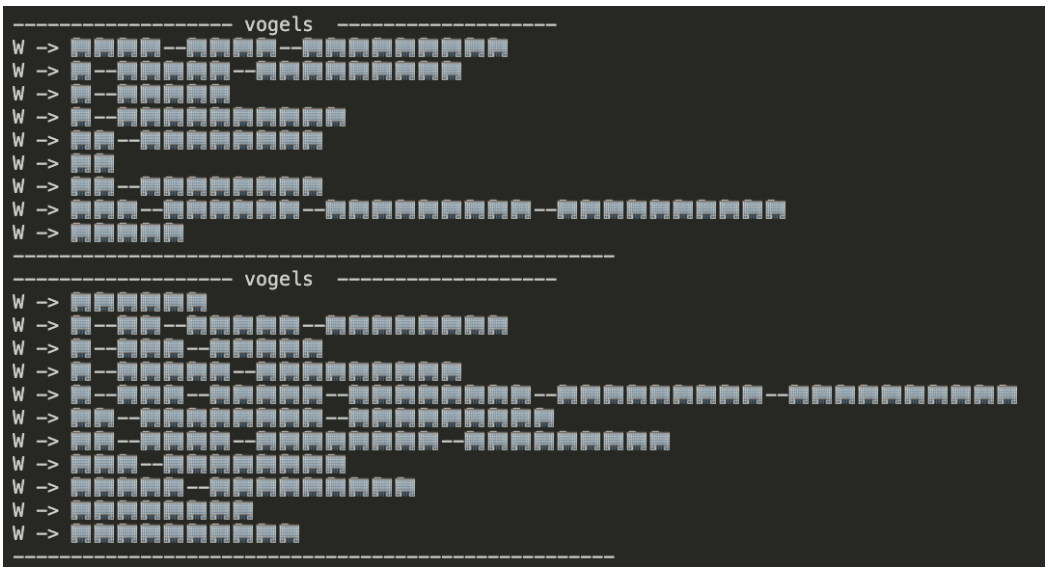



Рис. 18. Приклад розподілення задач алгоритмом апроксимації Фогеля без балансування

З огляду на рисунок видно, що є певне відхилення від оптимального розподілу в результатах роботи на середніх розмірах даних. При балансуванні ситуація покращується, що можна побачити на рис. 19, де W – працівник, а 🏢 – задача. Кількість послідовних 🏢 – час виконання задачі.

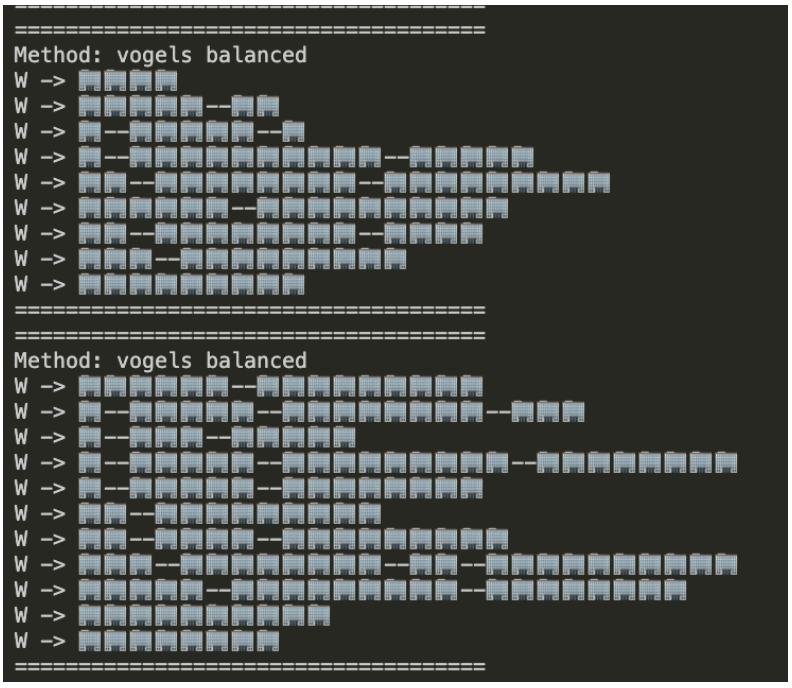



Рис. 19. Приклад розподілення задач алгоритмом апроксимації Фогеля з балансуванням

При роботі з більшими розмірами даних виникають більш вагомі перекоси без балансування, які вирішуються балансуванням. Це зображено на рис. 20 та рис. 21.

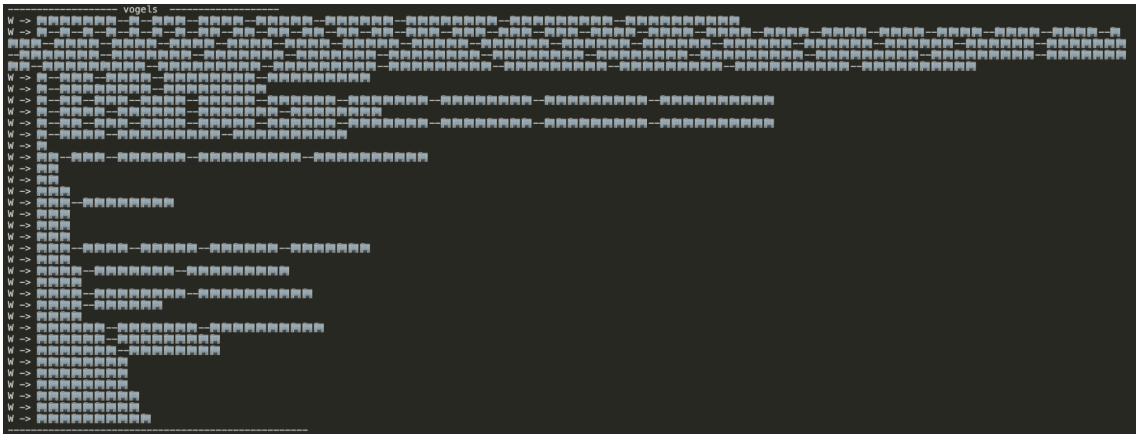


Рис. 20. Приклад розподілення задач алгоритмом апроксимації Фогеля без балансування з великою кількістю ОР

Ці перекоси успішно зникають і результат стає більш збалансований.

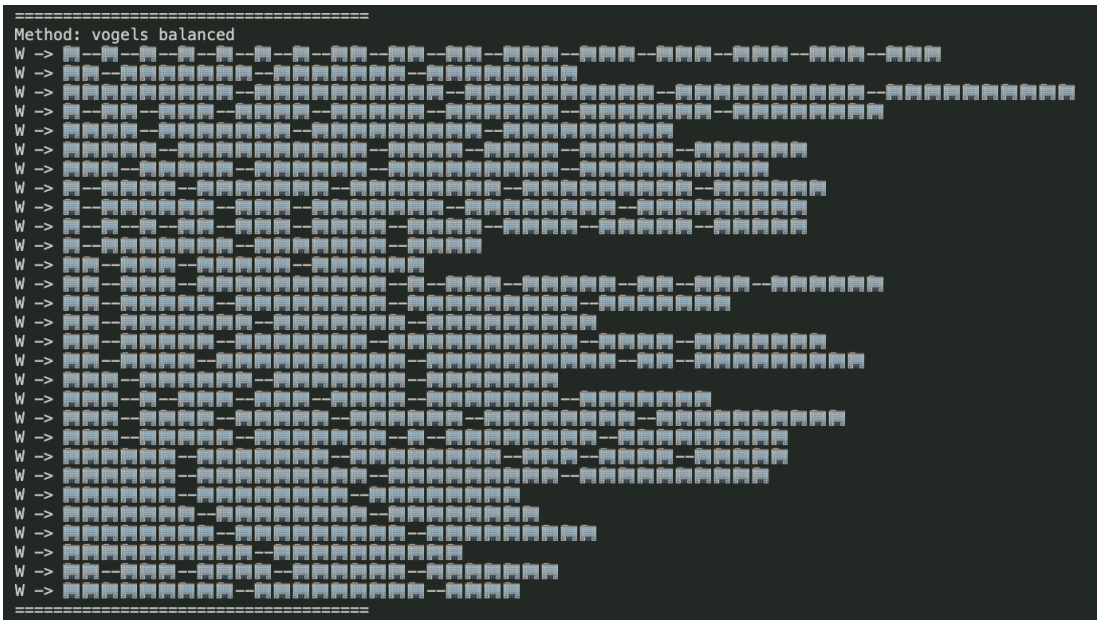


Рис. 21. Приклад розподілення задач алгоритмом апроксимації Фогеля з балансуванням з великою кількістю ОР

Даний результат на рис. 21 також є злегка розбалансованим, але при цьому процесі також було враховано можливість виконавцем зробити певну задачу, що призводить до того, що хтось з виконавців не може отримати достатньо задач.

## 4.4. Порівняння отриманих результатів з існуючими програмними аналогами

Переходимо до порівняльного аналізу розробленого алгоритму розподілення задач з існуючими аналогами, такими як OptaPlanner, Google OR-Tools та CPLEX Optimization Studio. Кожен з цих інструментів має свої унікальні характеристики, але вимагає інтеграції з відповідними SDK або вивчення специфічного доменного API.

OptaPlanner є потужним інструментом для розподілу задач, здатним опрацьовувати до 10,000 одиниць роботи (OP) [9]. У випадку запропонованого програмного рішення, система ефективно працює з наборами до 8,000 OP, що робить її конкурентоспроможною. На відміну від OptaPlanner, розроблена система не вимагає використання додаткових бібліотек або інтеграції, оскільки вона розроблена як незалежний консольний застосунок.

Google OR-Tools демонструє високу продуктивність і гнучкість у розподілі задач, пропонуючи різноманітні оптимізаційні алгоритми [10]. Цей інструмент може ефективно обробляти складні оптимізаційні задачі, але його використання вимагає більш глибокого розуміння специфіки оптимізаційних проблем та відповідного програмування.

CPLEX Optimization Studio, з іншого боку, використовує специфічну доменну мову і вимагає більш глибокого занурення в оптимізаційні методи. Це може створити бар'єр для користувачів, які не мають досвіду в роботі з такими системами.

У порівнянні з цими аналогами, розроблене програмне забезпечення вирізняється відсутністю необхідності у складній інтеграції чи вивченні

додаткових мов або інструментів. Хоча готові рішення можуть бути теоретично більш потужними у певних аспектах, розроблена система пропонує достатню ефективність та точність для вирішення поставлених задач, а також гнучкість і простоту у використанні, що робить її привабливою альтернативою для специфічних вимог користувачів.

## 4.5. Пропозиції для покращення системи

Нижче представлено кілька пропозицій щодо майбутніх оновлень:

1. Розвиток як SDK для різних мов програмування. Розробка алгоритму як гнучкого SDK, сумісного з різними мовами програмування, дозволяє його широке використання у різноманітних застосунках та системах.
2. Використання готових систем "під капотом". враховуючи результати порівняння швидкодії з аналогами, такими як OptaPlanner чи Google OR-Tools, можна розглянути можливість інтеграції цих систем в даний продукт.
3. Розробка користувацького інтерфейсу для роботи з задачами. Створення інтуїтивного графічного інтерфейсу користувача дозволить зробити систему доступною ширшому колу користувачів, які не знайомі з програмуванням;
4. Розширення можливостей системи. Поступове додавання нових функцій та інструментів, наприклад, додавання залежностей між задачами, залежність часу виконання задачі від виконавця, збільшення ліміту на кількість ОР, з якими може працювати система, інтеграція з іншими інструментами;
5. Постійне вдосконалення алгоритмів. Вдосконалення алгоритмів є необхідним, оскільки поточна реалізація демонструє дещо проблемні результати на великих наборах даних.

Ці пропозиції дозволять зробити розроблену систему більш адаптивною та гнучкою для задоволення потреб ринку.

## 4.6. Висновки до розділу

Цей розділ присвячений аналізу результатів, отриманих з реалізованого алгоритму розподілення задач, та їх порівняння з існуючими програмними рішеннями в цій сфері, такими як OptaPlanner, Google OR-Tools та CPLEX Optimization Studio. У процесі дослідження були виявлені ключові переваги та обмеження підходу з дисертації у порівнянні з цими системами. Особливу увагу було приділено аналізу ефективності алгоритмів та їх практичного застосування в різних сценаріях.

Результати порівняння показали, що розроблена система забезпечує конкурентоспроможні показники ефективності та гнучкості, особливо в контексті обробки середнього обсягу задач. Хоча деякі з розглянутих аналогів пропонують більш розширені можливості для великомасштабних операцій, розроблений алгоритм вирізняється своєю простотою та легкістю інтеграції у різні системи.

Також у розділі були сформульовані пропозиції щодо покращення та розвитку системи, зокрема розробка як SDK для різних мов програмування, інтеграція з готовими системами для підвищення продуктивності, розробка користувацького інтерфейсу та постійне вдосконалення алгоритмів. Ці пропозиції допоможуть розширити функціональність та спектр застосування системи, зробивши її більш привабливою для широкого кола користувачів.

Загалом, аналіз показав, що розроблена система має високий потенціал для подальшого розвитку і може бути ефективно використана у багатьох сферах, де необхідно оптимізувати розподіл задач. В майбутньому планується продовження роботи над проектом для вдосконалення його характеристик та покриття більш широкого спектру вимог користувачів.

# 5. ПОБУДОВА БІЗНЕС-МОДЕЛІ

## 5.1. Опис проблеми

У сучасних умовах розробки та впровадження алгоритмів, виникає складна проблема оптимального розподілу абстрактних завдань між визначеними виконавцями. Цей процес включає в себе аналіз та планування, засноване на різноманітних параметрах, таких як складність завдань, час на їх виконання, та можливі взаємозалежності між завданнями. Розробка алгоритмів, які ефективно керують цим процесом, вимагає глибокого розуміння теоретичних основ алгоритміки та їх практичного застосування.

Основна складність полягає у створенні алгоритмів, здатних ефективно розподіляти завдання, оптимізуючи використання часу та ресурсів виконавців. Це включає в себе не лише вибір відповідного виконавця для кожного завдання, але й врахування складності та пріоритетності завдань, а також потенційних залежностей між ними. Розробка таких алгоритмів має велике значення, оскільки вона впливає на загальну продуктивність та ефективність робочого процесу.

Дерево проблем зображене на рис. 22.

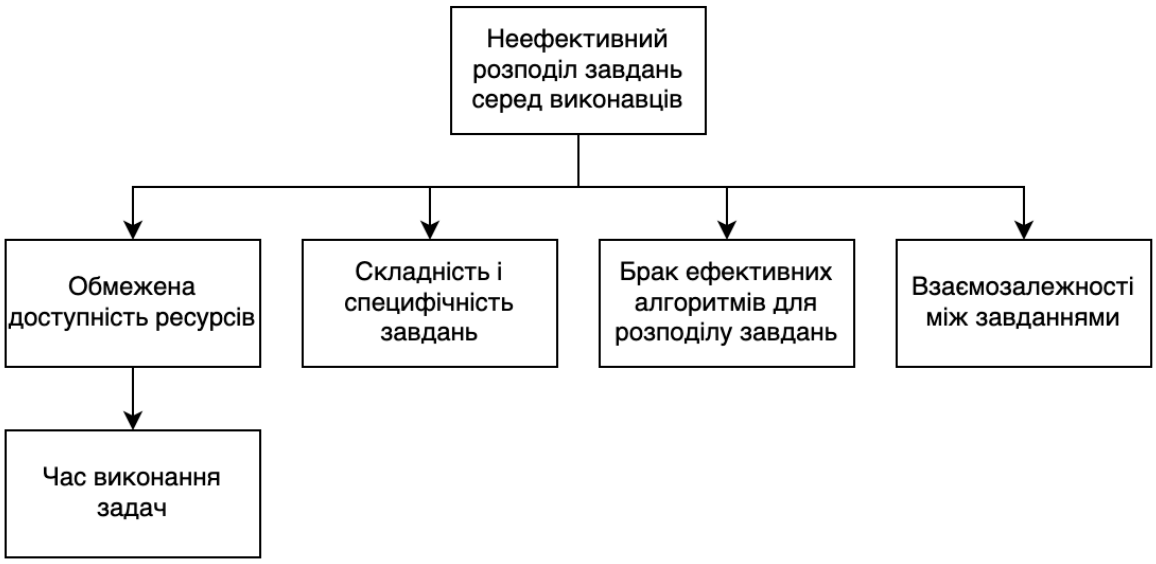



Рис. 22. Дерево проблем процесу розподілу задач між виконавцями

Метою цієї магістерської роботи є розробка алгоритмічних методів та програмного забезпечення для оптимізації процесу розподілу завдань. Основна увага приділяється вирішенню проблеми ефективного алгоритмічного планування в умовах, де кількість завдань та виконавців є значною, а також де потрібно враховувати різні обмеження та умови.

## 5.2. Аналіз зацікавлених сторін

Аналіз певних зацікавлених сторін можна переглянути у таблиці 6.

Таблиця 6

Зацікавлені сторони проекту та їхні інтереси.

| № | Зацікавлена сторона | Сфера зацікавленості |
|---|---|---|
| 1 | Розробники системи | Розробка, оптимізація, тестування програмного забезпечення |
| 2 | Представники бізнесу | Вимоги, стратегія, відслідковуваність, фінансова ефективність |
| 4 | Аналітики даних | Аналіз ефективності розподілу завдань, прогнозування навантаження |
| 5 | IT адміністратори | Інтеграція системи з іншими IT рішеннями, підтримка інфраструктури |
| 6 | Спеціалісти з оптимізації | Розробка алгоритмів оптимізації, підвищення ефективності розподілу завдань |
| 7 | Клієнти та замовники | Інтерес у результативності та оптимальності виконання проектів |
| 8 | Провайдери систем відстежування задач | Інтеграція з існуючими рішеннями для відстежування завдань, обмін даними |

У контексті розробки систем розподілу завдань, роль розробників системи полягає у створенні та оптимізації програмного забезпечення. Вони зосереджені на технічних аспектах і якості кінцевого продукту. Представники бізнесу зацікавлені в стратегічній відповідності системи бізнес цілям та її вартості. Аналітики даних використовують систему для виявлення інсайтів, які покращують розподіл ресурсів та завдань. Адміністратори відповідають за інтеграцію системи в існуючу інфраструктуру та її безперебійну роботу. Спеціалісти з оптимізації розробляють алгоритми, які забезпечують ефективне виконання завдань. В кінці, клієнти та замовники шукають в системі надійності та високої продуктивності проектів, а провайдери систем відстежування задач та конкурентні розробники відстежують інтеграцію та новітні тренди у сфері управління завданнями.

## 5.3. Інноваційне програмне рішення

Програмне рішення, яке розробляється в рамках цього дослідження, є консольним додатком, орієнтованим на ефективний розподіл завдань між виконавцями. Цей інструмент забезпечує функціональність для оптимізації процесів, враховуючи кваліфікацію робітників, обмеженість ресурсів та терміни виконання задач.

Ключовим елементом консольного додатку є алгоритм оптимізації, який розподіляє задачі між робітниками. Користувачі можуть вводити дані про задачі та доступних робітників через інтерфейс командного рядку, включаючи деталі задач, їх тривалість та список робітників, які можуть їх виконати. Після цього додаток генерує оптимальний план роботи.

Наразі інтеграція з іншими IT системами і проектними інструментами не реалізована, але планується для майбутнього розвитку через SDK та API. Це дозволить згодом об'єднувати зусилля з іншими інструментами та платформами, полегшуючи впровадження оптимізації в існуючі робочі процеси.

Проект акцентує увагу на гнучкості і масштабованості, забезпечуючи можливість адаптації до широкого спектру бізнес сценаріїв та вимог, а також на підвищення продуктивності великої кількості робітників та задач, які їм треба виконати, мінімізуючи простої та оптимізуючи загальний час виконання роботи.

## 5.4. Комерційний програмний продукт. Конкурентні переваги програмного продукту

Програмний продукт, розроблений в рамках даної роботи, пропонує оновлений підхід до оптимізації розподілу завдань, використовуючи алгоритмічне планування, що зводить до мінімуму людський фактор та підвищує точність і ефективність розподілення задач. Цей консольний додаток спрямований на максимізацію продуктивності виконавців та оптимізацію використання ресурсів.

Комерційно, продукт призначений для підвищення ефективності виробничих процесів, забезпечуючи керівникам проектів інструмент для більш обґрунтованого розподілу завдань. Подальша інтеграція з різними IT системами забезпечить гладке впровадження вже відомих інструментів у робочі процеси компаній.

Основні конкурентні переваги програмного продукту:

- автоматизований підхід – використання розширеного алгоритму для розподілу завдань дозволяє підвищити точність планування та зменшити вплив людського фактору;
- підвищення продуктивності – ефективне розподілення роботи забезпечує краще використання часу та ресурсів, знижуючи простої і оптимізуючи робочі навантаження;
- гнучкість і масштабованість – продукт легко адаптується під різноманітні умови виконання та розміри проектів, забезпечуючи рішення для широкого спектру бізнес задач.

У майбутньому планується розширення можливостей продукту через розробку SDK та API, що надасть користувачам можливість інтегрувати алгоритм оптимізації безпосередньо в їхні рішення.

## 5.5. Ринок аналогічних програмних рішень

У сфері управління проектами і розподілу задач існує велика кількість інструментів, які пропонують різні підходи до оптимізації робочих процесів. Аналіз ринку аналогічних програмних рішень дозволяє визначити сильні та слабкі сторони конкурентів і відповідно адаптувати стратегію власного продукту для забезпечення конкурентних переваг.

OptaPlanner, Google OR-Tools та CPLEX Optimization Studio є провідними рішеннями на ринку, які надають широкий спектр оптимізаційних інструментів для рішення задач розподілу та планування. Кожен з цих інструментів має свої особливості, які роблять їх ідеальними для певних варіантів застосування. OptaPlanner є відмінним рішенням для складних планувальних задач, Google OR-Tools надає відкритий набір інструментів для різних типів оптимізацій, а CPLEX Optimization Studio від IBM пропонує інтегровані рішення для великих та складних моделей оптимізації.

Розроблена система зосереджена на специфічній задачі оптимізації розподілу задач, надаючи просту, але не менш ефективну альтернативу до уже готових систем зі своїми базами користувачів.

## 5.6. Унікальна ціннісна пропозиція

Цей інструмент надає цінність зацікавленим сторонам, таким як IT адміністратори та спеціалісти з оптимізації, які шукають швидке, надійне та легко інтегроване рішення. Для клієнтів, які займаються аналізом даних, він пропонує можливість точного прогнозування навантаження та ефективності виконання завдань.

На відміну від інших існуючих рішень на ринку, які зазвичай пропонують широкий спектр управлінських інструментів, розроблений додаток зосереджений на специфіці розподілу завдань, пропонуючи конкретну функціональність без зайвого навантаження функціями, що рідко використовуються.

Основна УЦП продукту полягає в наступному:

- ефективність алгоритмів – продукт включає розроблений алгоритм оптимізації, який мінімізує час виконання завдань та забезпечує оптимальне використання ресурсів;
- гнучкість інтеграції – можливість інтегрувати продукт у вже існуючі робочі процеси через SDK, забезпечуючи плавне введення в експлуатацію без потреби в розробці власного API.

Ці аспекти визначають УЦП як потужний інструмент для бізнесів, що прагнуть оптимізувати свої робочі процеси у різних сферах використання. Дане рішення може бути модифікованих і використаним не тільки в домені проектного менеджменту, а і у задачах пріоритизації та оптимізації роботи операційних та пошукових систем, систем роботи з мережами, тощо.

## 5.7. Потоки доходів. Структура витрат

З огляду на ймовірні способи використання системи було описано певні потенційні потоки доходів:

1. Модель оплати за використання – клієнти оплачують за кожне використання алгоритму. Наприклад, оплата за кожні 5 разів використання. Це гнучка модель, яка підійде для клієнтів із змінним обсягом завдань.
2. Підписка на пакети запитів – різні пакети підписки (наприклад, 100, 500, 1000 запитів на місяць) забезпечують клієнтам гнучкість у виборі залежно від їх потреб. Ці запити можна буде використати як у SDK через API так і через веб інтерфейс.

3. Преміум підписка – для більших організацій або вимогливих проектів можна запропонувати преміум підписку, яка включає додаткові функції, такі як розширений аналіз даних, індивідуальні налаштування алгоритму, пріоритетна технічна підтримка тощо.

Структура витрат має такий вигляд:

- заробітна плата – оплата праці розробників, аналітиків, тестувальників, маркетологів, та технічної підтримки;
- технічні витрати – оплата за хостинг, обслуговування серверів, ліцензії розробника, та інші технічні ресурси;
- маркетинг та реклама – витрати на онлайн рекламу, участь у конференціях, створення маркетингових матеріалів;
- адміністративні витрати – витрати на оренду офісу, комунальні послуги, податки, та інші загальноадміністративні витрати;
- податки.

Для того, щоб продукт був конкурентоспроможним та привабливим для потенційних клієнтів, можна також розглянути такі ідеї:

- безкоштовна пробна версія – надати клієнтам можливість випробувати продукт безкоштовно з обмеженими функціями або кількістю запитів.
- гнучкі цінові пропозиції – надати можливість користувачам налаштовувати підписку в залежності від їх потреб та масштабу проекту;
- спеціальні знижки для стартапів або освітніх установ – залучити нові сегменти ринку, пропонуючи знижки або спеціальні умови для стартапів, студентів або освітніх організацій.

## 5.8. Канва бізнес-моделі стартапу

Ціль канви бізнес моделі – описати продукт чи проаналізувати бізнес-модель, яка вже використовується, з позицій її ефективності та можливості

до розвитку. Враховуючи усі вищезгадані пункти, можна сформувати таку бізнес-модель у вигляді lean canvas:

- проблема – неефективний розподіл завдань у великих проектах, що вимагає значних людських ресурсів та часу;
- рішення – консольний додаток, який за допомогою спеціалізованих алгоритмів оптимізує розподіл завдань, мінімізуючи час виконання та збільшуючи загальну продуктивність;
- унікальна ціннісна пропозиція – ефективність в розподілі завдань через використання оптимізаційних алгоритмів, мінімалістичний інтерфейс для простоти користування у майбутньому, гнучкість інтеграції через SDK;
- прихована перевага – автоматизація та оптимізація процесу розподілу завдань, зниження витрат часу та ресурсів на планування;
- споживачі – IT адміністратори, спеціалісти з оптимізації, аналітики даних, бізнес аналітики, менеджери великих проектів;
- ключові метрики – кількість активних користувачів, задоволеність клієнтів, ефективність розподілу завдань (зменшення часу на виконання), впровадження в нові компанії;
- канали – прямі продажі B2B, партнерства з IT компаніями, онлайн маркетинг, відвідування професійних заходів і конференцій, вебінари та демонстрації продукту;
- структура витрат – розробка та підтримка продукту, маркетинг та реклама, витрати на хостинг та інфраструктуру, витрати на персонал (розробники, тестувальники, маркетологи);
- потоки доходів – підписка на використання програми, спеціалізовані рішення на замовлення для великих клієнтів.

## 5.9. Висновки до розділу

У рамках даного розділу було проведено аналіз проблематики ефективного розподілу завдань та визначено ключові аспекти розробки програмного рішення. Зацікавлені сторони та їх вимоги були ретельно вивчені, що дозволило сформувати чітку картину потреб ринку. Аналіз конкурентного середовища показав, що хоча ринок оптимізаційних інструментів є насиченим, запропоноване програмне рішення має потенціал зайняти свою нішу завдяки своїм унікальним характеристикам та фокусу на конкретних задачах оптимізації.

Унікальна ціннісна пропозиція розробленого продукту полягає в точному і гнучкому алгоритмі розподілу завдань.. Це рішення відповідає сучасним вимогам ринку, де швидкість та точність є ключовими параметрами успіху.

Розглянуто різні моделі монетизації продукту, включаючи оплату за використання, підписки на пакети запитів. Такий підхід дозволить задовольнити потреби різних категорій користувачів і забезпечити гнучкість у виборі оптимального плану використання продукту.

Структура витрат була продумана з урахуванням всіх необхідних аспектів функціонування та підтримки продукту, включаючи заробітну плату, технічні витрати, маркетинг та адміністративні витрати.

Результатом цього дослідження є комплексна бізнес-модель, яка враховує ключові елементи успішної комерціалізації програмного продукту: визначення проблеми, розробка ефективного рішення, аналіз зацікавлених сторін, розуміння конкурентного середовища, формування унікальної ціннісної пропозиції та ретельне планування доходів та витрат. Все це створює міцну основу для успішного впровадження та розвитку продукту на ринку

# ВИСНОВКИ

У цьому дипломному проекті було проведено аналіз основних методів та алгоритмів планування порядку виконання задач, що визначило ключові аспекти та властивості запропонованого програмного рішення такі як простота використання та зосередженість на одній конкретній задачі. З аналізу основним алгоритмів було вирішено, що робота буде фокусуватися на алгоритмі, який динамічно обиратиме реалізацію і залежності від розміру вхідних даних.

При модифікації нового алгоритму було обрано рішення розбити його на 3 частини: повний перебір, апроксимація Фогеля та балансування результатів. Це було зроблено через обмеження у часі роботи, які стосуються кожного окремого етапу алгоритму. Зокрема повний перебір є дуже малоефективним починаючи з порівняно невеликої кількості даних, оскільки має факторіальну складність. Власний алгоритм балансування результатів апроксимації Фогеля запропоновано з урахуванням специфічності вхідних даних та потенційних обмежень, які накладаються на алгоритм потенціалів, який часто використовується в парі з алгоритмом Фогеля.

Для розробки було визначено функціональні та нефункціональні вимоги, проведено аналіз обраних інструментів розробки на базі JavaScript та побудовано модульну архітектуру системи.

Згідно з аналізом отриманих результатів алгоритм працює на високому рівні якості і досягає порівняно рівномірного розподілення задач між виконавцями за досить малий час. Розподілення 160 задач між 50 робітниками займає в середньому 140 мс і є достатньо рівномірним, що вирішує одну з основних задач: мінімізацію часу простою виконавців певних робіт. Така швидкодія є конкурентоспроможною серед аналогів.

З огляду на вимоги, аналіз конкурентів та ринку було проведено аналіз проблематики та зацікавлених сторін. Результатом є запропоновані потоки доходів та витрат і побудовано бізнес-модель.

У підсумку було проведено комплексний аналіз проблематики, побудовано бізнес-модель та запропоновано й розроблено програмне рішення, що відповідає вимогам ринку та має великий потенціал для розвитку.

# СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ


1. *Nahar, J & Rusyaman, E & Putri, Verentia Eka*. (2018). Application of improved Vogel's approximation method in minimization of rice distribution costs of Perum BULOG. IOP Conference Series: Materials Science and Engineering. 332. 012027. 10.1088/1757-899X/332/1/012027.
2. *Zoran G*. JOB ASSIGNMENT PROBLEM SOLVED BY NEURO-FUZZY INFERENCE SYSTEM / G. Zoran, K. Ivan, A. Sime. – FON University, Bul. Vojvodina, bb, Skopje, Macedonia tel. 389 22 445-592, fax 389 22 445-550, gacovski@mt.net.mk
3. Liu, Linzhong, and Xin Gao. "Fuzzy Weighted Equilibrium Multi-Job Assignment Problem and Genetic Algorithm." Applied Mathematical Modelling 33, no. 10 (2009): 3926–35. doi:10.1016/J.APM.2009.01.014.
4. Russel Approximation Method And Vogel's Approximation Method In Solving Transport Problem. // International Journal of Informatics and Computer Science (The IJICS) ISSN 2548-8384 (online) Vol 1. – 2017. – №1. – С. 1–6.
5. Транспортна задача [Електронний ресурс]: навч. посіб. для студ. спеціальності 121 «Інженерія програмного забезпечення», спеціалізації «Програмне забезпечення розподілених систем», «Програмне забезпечення Web-технологій та мобільних пристроїв»/ КПІ ім. Ігоря Сікорського ; уклад.: О. К. Молодід. – Електронні текстові дані (1 файл:1,06 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 37 с.
6. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en.
7. Jest · Delightful JavaScript Testing [Електронний ресурс] – Режим доступу до ресурсу: https://jestjs.io/uk/.
8. Faker | Faker [Електронний ресурс] – Режим доступу до ресурсу: https://fakerjs.dev/.

9. TypeScript: JavaScript With Syntax For Types. [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/.
10. OptaPlanner - Task assigning :: Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.optaplanner.org/docs/optaplanner/latest/use-cases-and-examples/task-assigning/task-assigning.html#taskAssigning.
11. Assignment with Task Sizes | OR-Tools | Google for Developers [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/optimization/assignment/assignment_cp#python.
12. Вікіпедія – Алгоритм Крускала [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D1%80%D1%83%D1%81%D0%BA%D0%B0%D0%BB%D0%B0#%D0%9E%D1%86%D1%96%D0%BD%D0%BA%D0%B0_%D1%81%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%96.
13. Вікіпедія – Алгоритм пошуку A* [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%BF%D0%BE%D1%88%D1%83%D0%BA%D1%83_A*.
14. А. М. Господінов. ГЕНЕТИЧНИЙ АЛГОРИТМ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ КОМІВОЯЖЕРА / А. М. Господінов, С. А. Смирнов. – С. 19–21.
15. Вікіпедія – "Мінімальне кістякове дерево." [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9C%D1%96%D0%BD%D1%96%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BA%D1%96%D1%81%D1%82%D1%8F%D0%BA%D0%BE%D0%B2%D0%B5_%D0

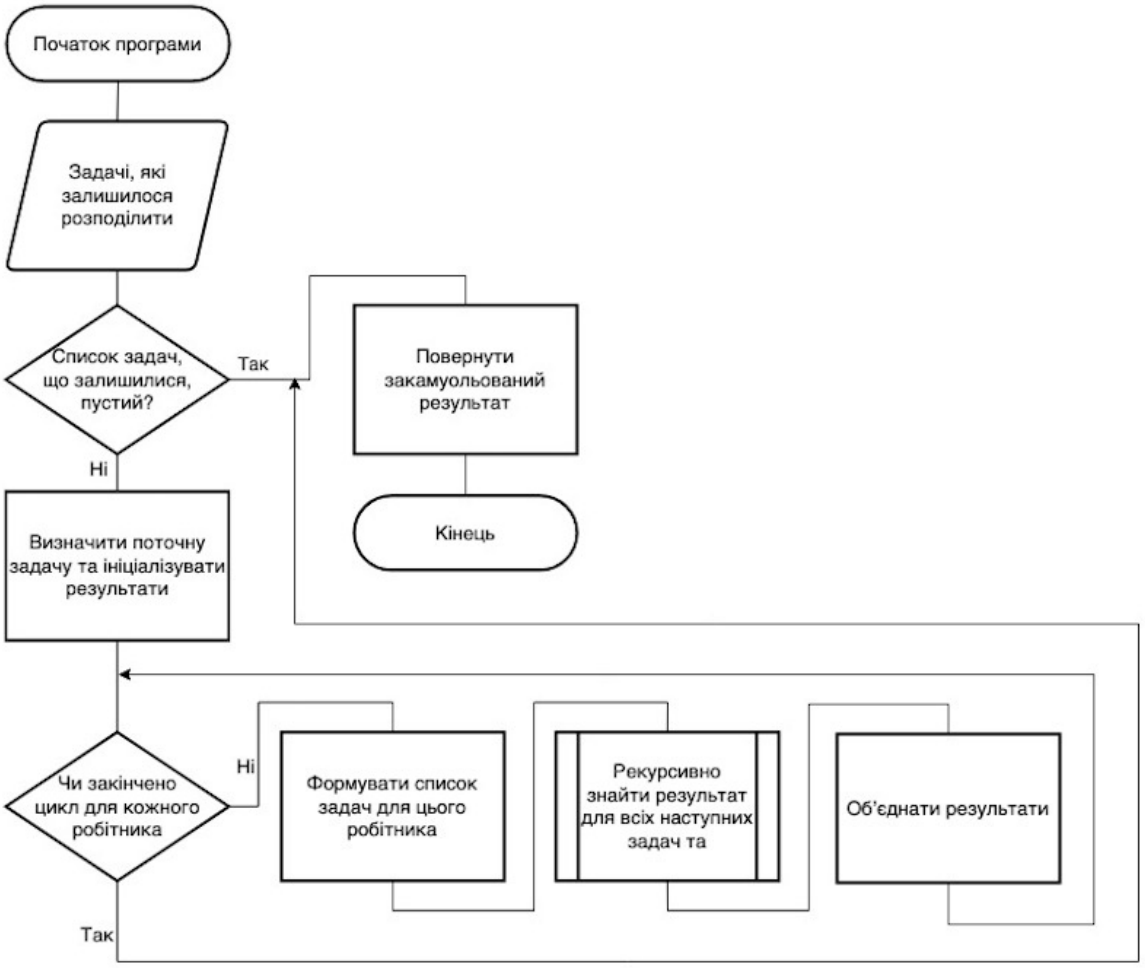
%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE

16. IBM ILOG CPLEX Optimization Studio [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/products/ilog-cplex-optimization-studio.
17. Хіцко Я.В., Коваль А.О. Спосіб та програмне забезпечення для організації та мінімізації часу виконання задач // Збірник тез XVI конференції молодих вчених «Прикладна математика та комп'ютинг», Київ. – 2023.- с. 617-622

# ДОДАТКИ

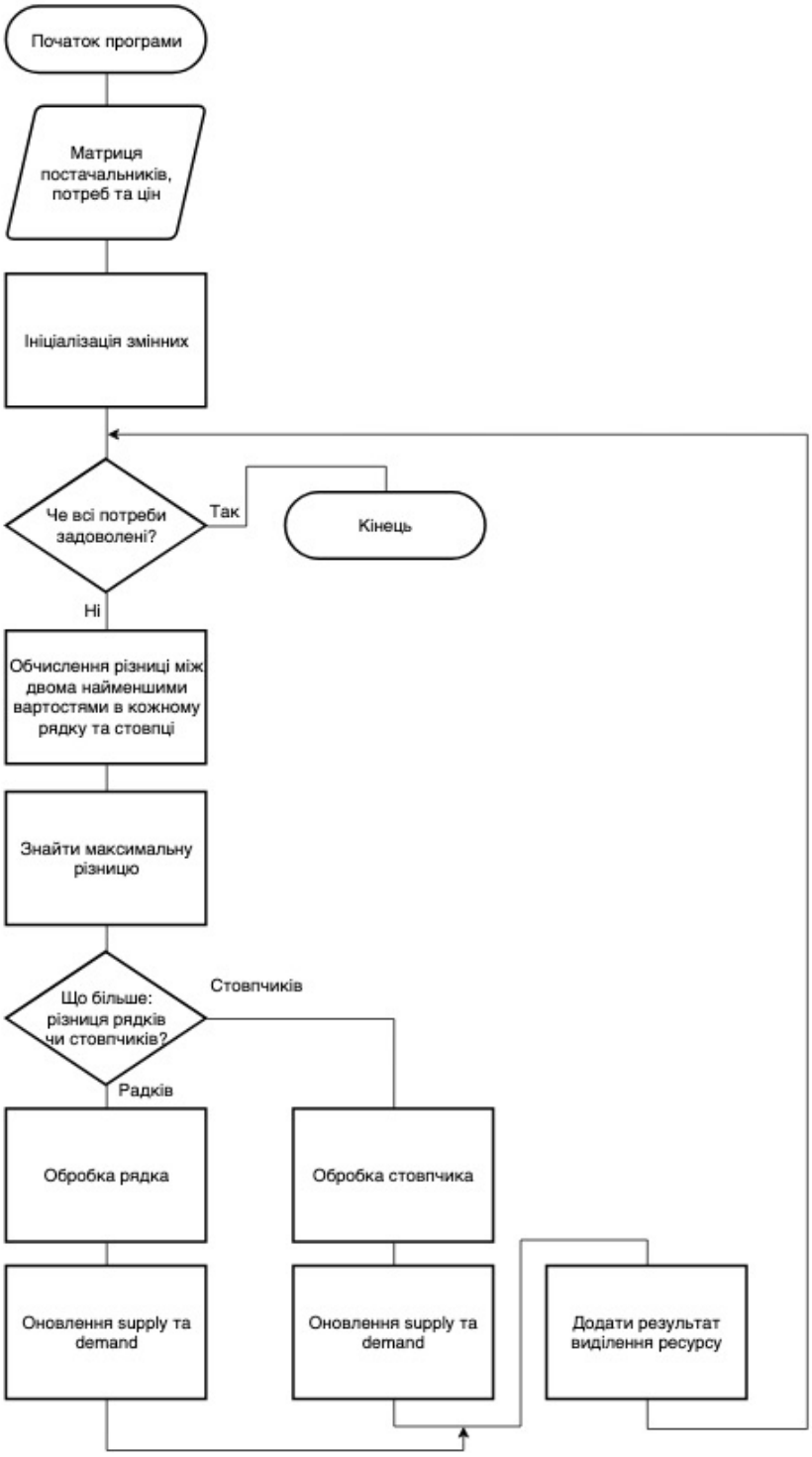
**Додаток 1**
**Копії графічних матеріалів**

Блок схема алгоритму перебору



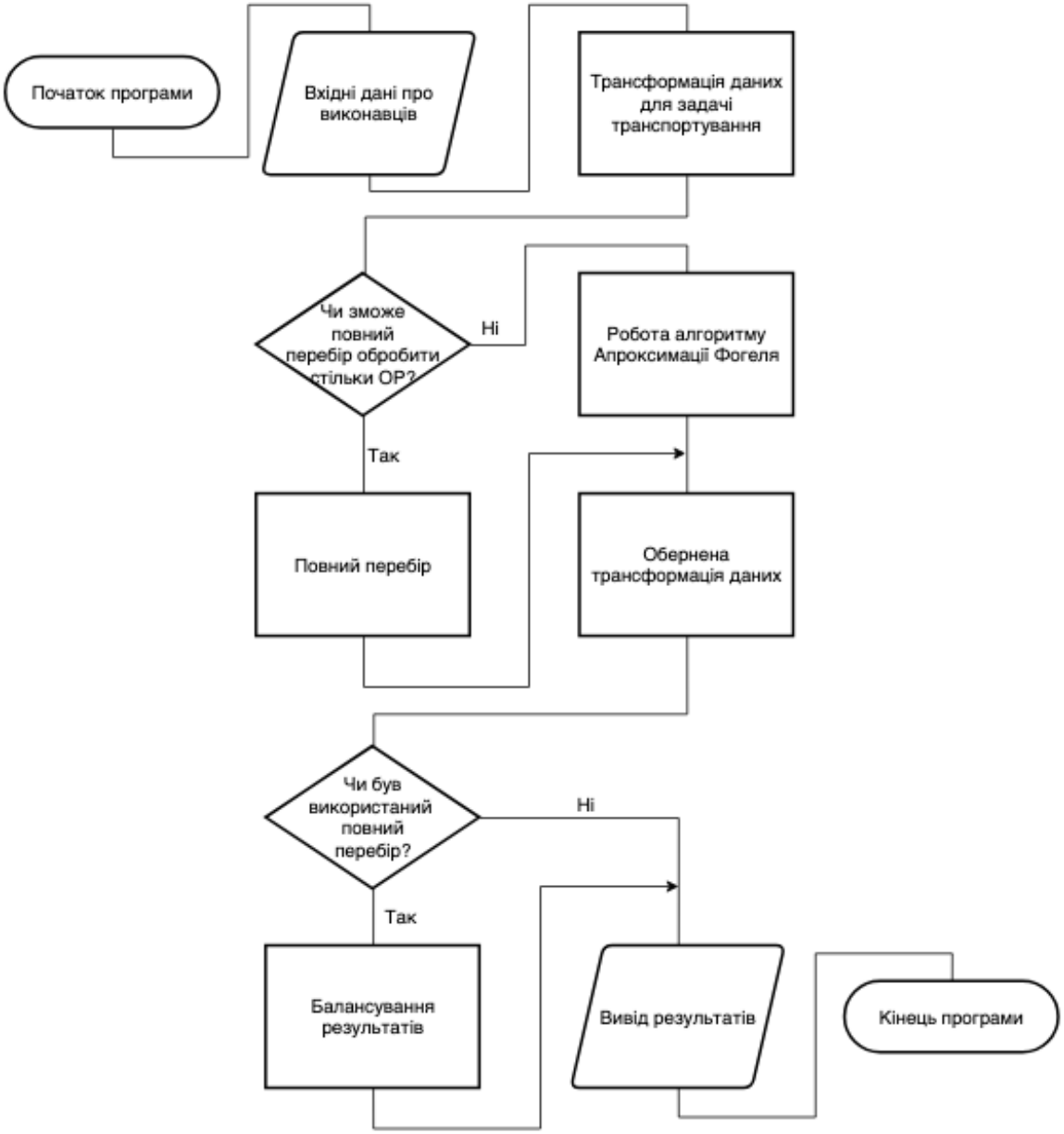

Коваль Андрій, КП-21мп

Блок схема алгоритму Фогеля



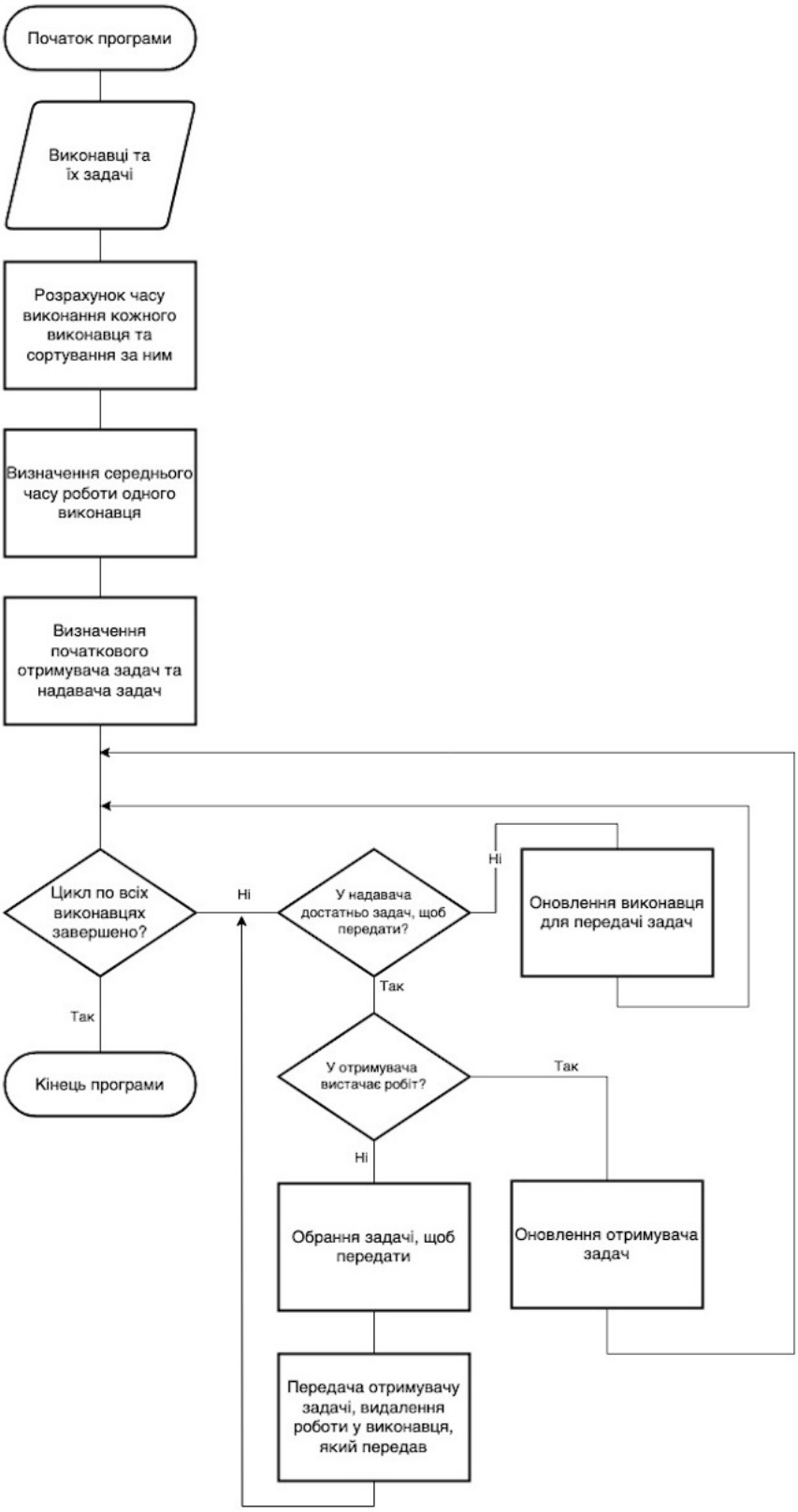

Коваль Андрій, КП-21мп

Блок схема алгоритму роботи програми



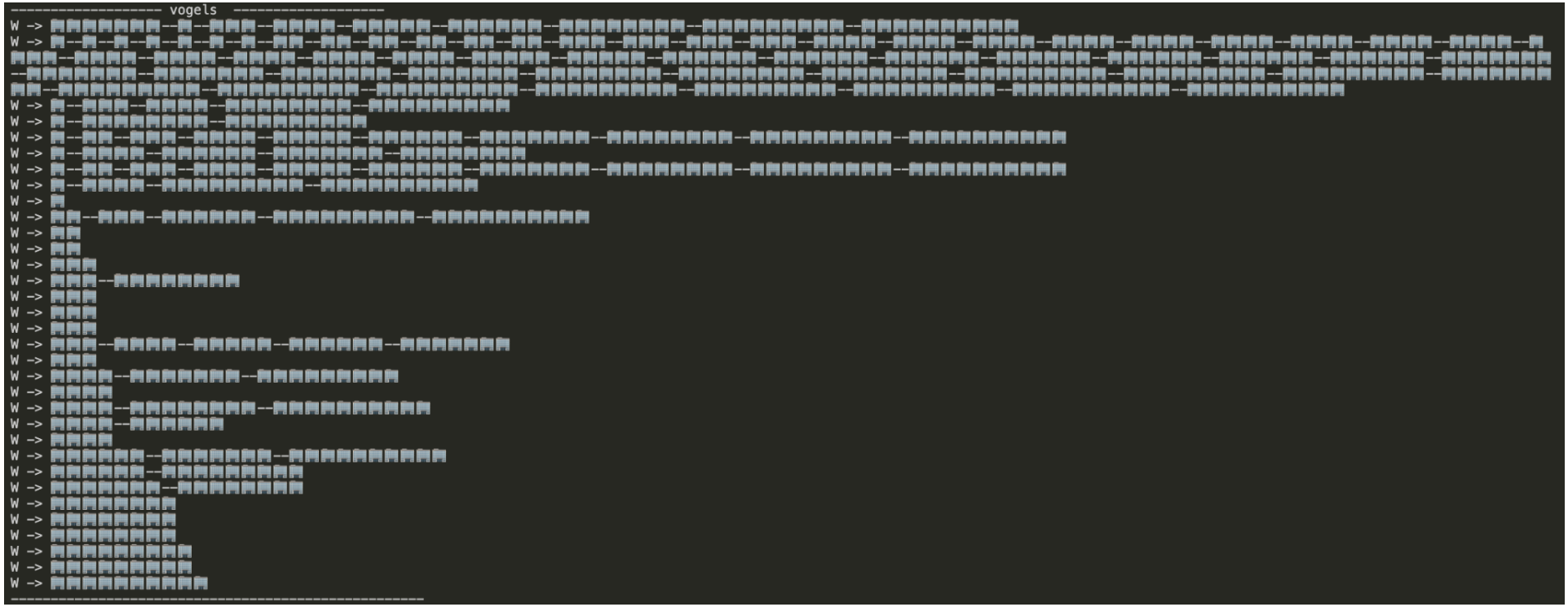

Коваль Андрій, КП-21мп

Блок схема алгоритму балансування результатів



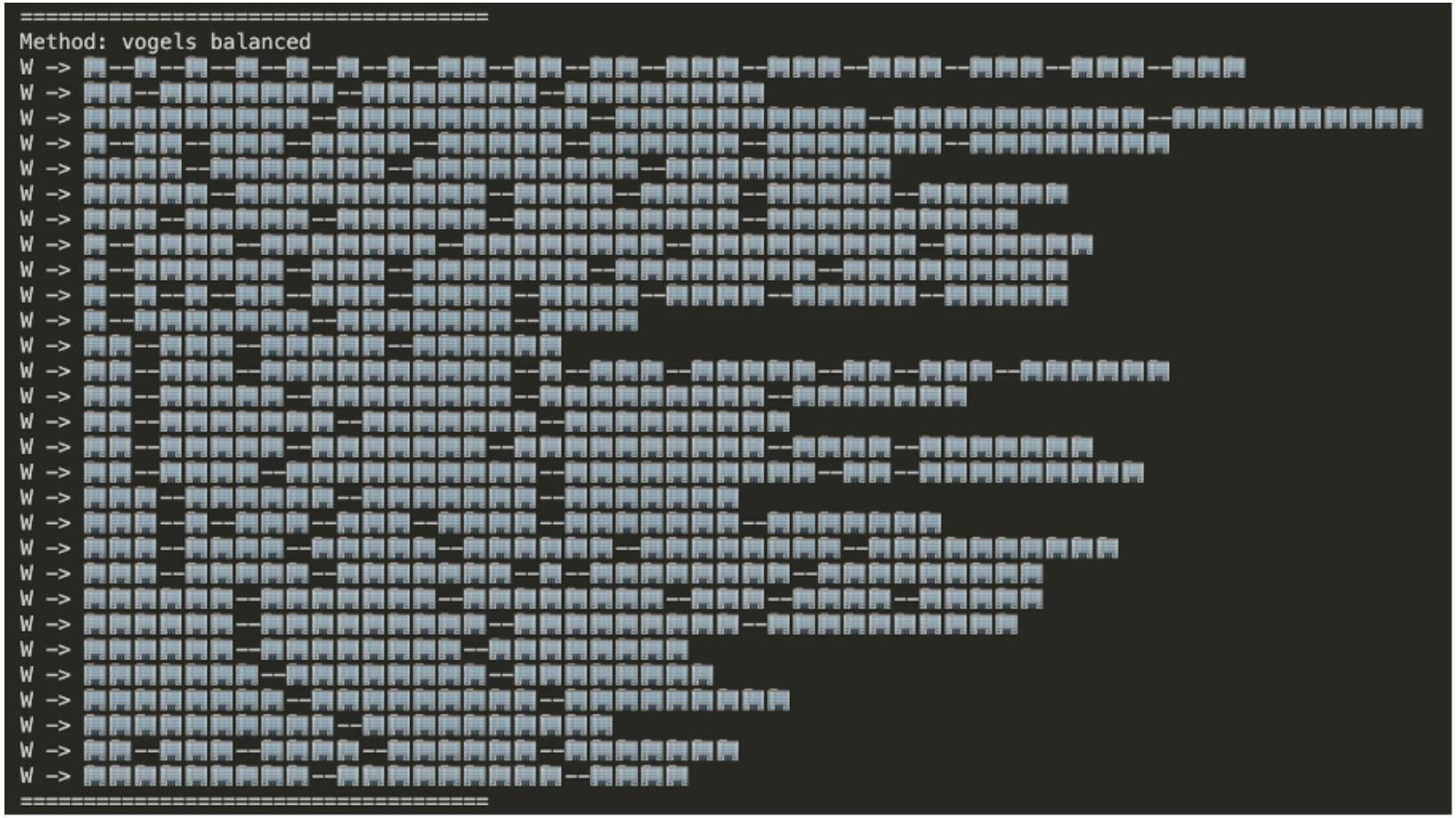

Коваль Андрій, КП-21мп

Приклад розподілення задач алгоритмом апроксимації Фогеля без балансування з великою кількістю ОР

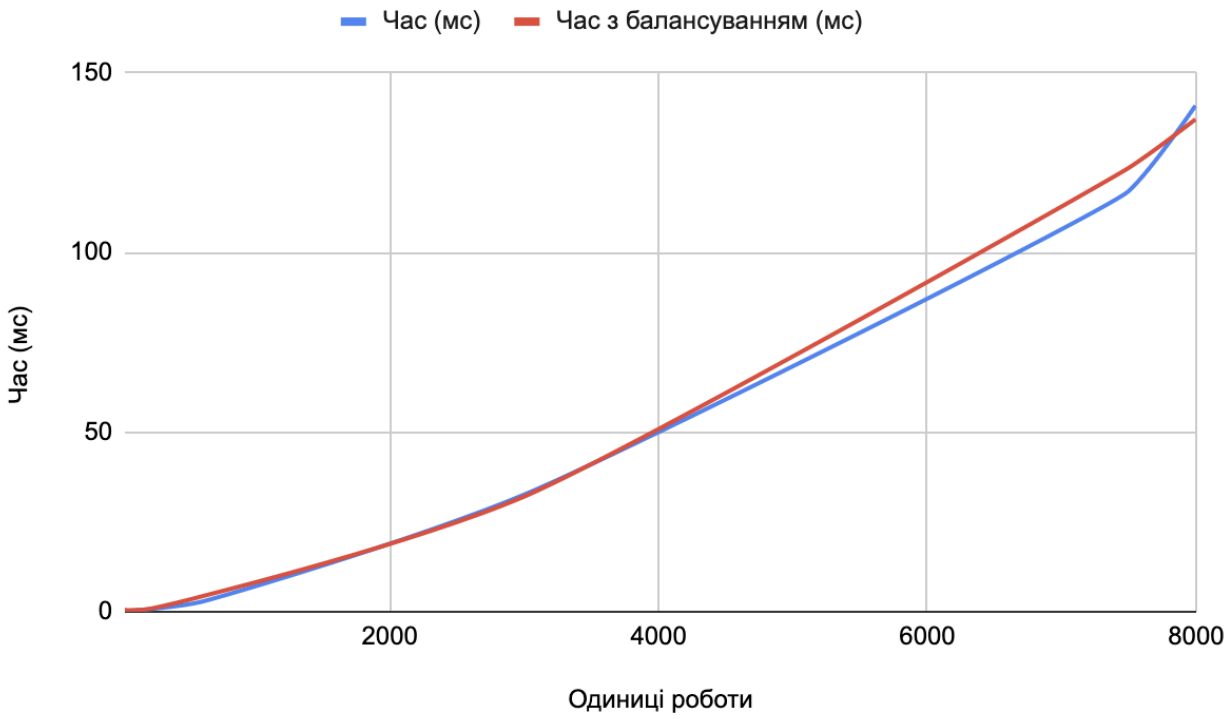



Коваль Андрій, КП-21мп

Приклад розподілення задач алгоритмом апроксимації Фогеля з балансуванням з великою кількістю ОР




Коваль Андрій, КП-21мп




Коваль Андрій, КП-21мп

**Додаток 2**
**Лістинг програми**

## vogelsApproximation.ts

```
import { Task, Worker } from '../../types';
import {
  TransportationProblemInput,
  TransportationProblemOutput,
} from './types';

export const vogelsApproximation = (
  input: TransportationProblemInput<Worker, Task>,
): TransportationProblemOutput<Worker, Task> => {
  const grid = input.costs;
  const supply = input.suppliers.map(supplier => supplier.supply);
  const demand = input.demands.map(demand => demand.demand);

  const res = vogelsApproximation2(grid, supply, demand);
  return {
      allocations: res.map(allocation => ({
      supplier: input.suppliers[allocation.supplierIndex],
      destination: input.demands[allocation.destinationIndex],
      allocatedAmount: allocation.value,
      })),
  };
};

type Grid = number[][];
type Supply = number[];
type Demand = number[];

type Allocation = {
  supplierIndex: number;
  destinationIndex: number;
  value: number;
};

const INF = Number.MAX_SAFE_INTEGER;

const vogelsApproximation2 = (
  grid: Grid,
  supply: Supply,
  demand: Demand,
): Allocation[] => {
  const n = grid.length;
  const allocations: Allocation[] = [];

  function findDiff(grid: Grid): [number[], number[]] {
      const rowDiff: number[] = [];
      const colDiff: number[] = [];

      for (let i = 0; i < grid.length; i++) {
      const arr = [...grid[i]];
      arr.sort((a, b) => a - b);
      rowDiff.push(arr[1] - arr[0]);
      }

      for (let col = 0; col < grid[0].length; col++) {
      const arr: number[] = [];
      for (let i = 0; i < grid.length; i++) {
      arr.push(grid[i][col]);
      }
      arr.sort((a, b) => a - b);
```

```
    colDiff.push(arr[1] - arr[0]);
    }

    return [rowDiff, colDiff];
}

while (Math.max(...demand) !== 0) {
    const [row, col] = findDiff(grid);
    const maxi1 = Math.max(...row);
    const maxi2 = Math.max(...col);

    if (maxi1 >= maxi2) {
    const ind = row.findIndex(val => val === maxi1);
    const mini1 = Math.min(...grid[ind]);
    const ind2 = grid[ind].findIndex(val => val === mini1);
    const mini2 = Math.min(supply[ind], demand[ind2]);

    allocations.push({
    supplierIndex: ind,
    destinationIndex: ind2,
    value: mini1,
    });
    supply[ind] -= mini2;
    demand[ind2] -= mini2;

    if (demand[ind2] === 0) {
    for (let r = 0; r < n; r++) {
          grid[r][ind2] = INF;
    }
    } else {
    grid[ind].fill(INF);
    }
    } else {
    const ind = col.findIndex(val => val === maxi2);
    let mini1 = INF;

    for (let j = 0; j < n; j++) {
    mini1 = Math.min(mini1, grid[j][ind]);
    }

    const ind2 = grid.findIndex(row => row[ind] === mini1);
    if (ind2 !== -1) {
    const mini2 = Math.min(supply[ind2], demand[ind]);

    allocations.push({
          supplierIndex: ind2,
          destinationIndex: ind,
          value: mini1,
    });
    supply[ind2] -= mini2;
    demand[ind] -= mini2;

    if (demand[ind] === 0) {
          for (let r = 0; r < n; r++) {
          grid[r][ind] = INF;
          }
    } else {
          grid[ind2].fill(INF);
    }
    }
    }
```

```
  }

  return allocations;
};
```

## lib/vogelsApproximation/mappers.ts

```
import { Input, Output, Task, Worker } from '../../types';
import {
  TransportationProblemInput,
  TransportationProblemOutput,
} from './types';

export const mapTasksInputToTransportationProblemInput = (
  input: Input,
): TransportationProblemInput<Worker, Task> => {
  const costs = input.workers.map(worker => {
      return input.tasks.map(task => {
      if (task.canBeDoneBy.includes(worker.id)) {
      return task.duration;
      }
      return Number.MAX_SAFE_INTEGER;
      });
  });

  const suppliers = input.workers.map(worker => ({
      id: worker.id,
      name: worker.name,
      supply: Number.MAX_SAFE_INTEGER,
  }));

  const demands = input.tasks.map(task => ({
      ...task,
      demand: task.duration,
  }));

  return {
      costs,
      suppliers,
      demands,
  };
};

export const mapTransportationProblemOutputToTasksOutput = (
  output: TransportationProblemOutput<Worker, Task>,
): Output => {
  const result: Record<Worker['id'], Task[]> = {};

  output?.allocations?.forEach(allocation => {
      if (!result[allocation.supplier.id]) {
      result[allocation.supplier.id] = [];
      }
      result[allocation.supplier.id].push({
      ...allocation.destination,
      id: allocation.destination.id,
      name: allocation.destination.name,
      });
```

```
  });

  return { result };
};
```

## lib/vogelsApproximation/types.ts

```
import { BaseItemMeta, Worker } from '../../types';

export type TransportationProblemInput<
  TSupplier extends BaseItemMeta = BaseItemMeta,
  TDestination extends BaseItemMeta = BaseItemMeta,
> = {
  suppliers: (TSupplier & { supply: number })[];
  demands: (TDestination & { demand: number })[];
  costs: number[][];
};

export type TransportationProblemOutput<
  TSupplier extends BaseItemMeta = BaseItemMeta,
  TDestination extends BaseItemMeta = BaseItemMeta,
> = {
  allocations: Array<{
      supplier: TSupplier;
      destination: TDestination;
      allocatedAmount: number;
  }>;
};
```

## lib/brutForce/brutForce.ts

```
import { Input, Output, Task } from '../../types';
import { getMaxDuration } from '../utils/duration';

/**
 * This method is theoretically correct for a small number of tasks and
workers
 *
 * It uses the combinations method to find all possible combinations of
tasks
 */
export const brutForce = (input: Input): Output => {
  const { workers, tasks } = input;
  const allPossibleResults = getAllPossibleTasksAssignments(tasks);
  const bestResult = getShortestMaximalDuration(
      allPossibleResults,
  );
  return {
      result: bestResult,
  };
};

const getShortestMaximalDuration = (
  allPossibleTaskAssignments: Record<string, Task[]>[],
): Record<string, Task[]> => {
  return allPossibleTaskAssignments.reduce((shortest, current) => {
```

```
      const currentMaxDuration = getMaxDuration(current);
      const shortestMaxDuration = getMaxDuration(shortest);
      return currentMaxDuration < shortestMaxDuration ? current :
shortest;
  }, allPossibleTaskAssignments[0]);
};

const getAllPossibleTasksAssignments = (
  remainingTasks: Task[],
  currentAssignment: Record<string, Task[]> = {},
): Record<string, Task[]>[] => {
  if (remainingTasks.length === 0) {
      return [currentAssignment];
  }

  const task = remainingTasks[0];
  let results: Record<string, Task[]>[] = [];

  for (const workerId of task.canBeDoneBy) {
      const tasksForWorker = [...(currentAssignment[workerId] || []),
task];

      const newAssignments = getAllPossibleTasksAssignments(
      remainingTasks.slice(1),
      {
      ...currentAssignment,
      [workerId]: tasksForWorker,
      },
      );
      results = results.concat(newAssignments);
  }

  return results;
};
```

## lib/balance/balance.ts

```
import { Output, Task } from '../../types';
import { getDuration } from '../utils/duration';
import { TransportationProblemOutput } from
'../vogelsApproximation/types';

type WorkerID = string;

/**
 * This method is intended to balance the output of the 2 first
algorithms.
 *
 * This is mostly heuristic alg.
 *
 * This is non pure function
 */
export const balance = (output: Output, numberOfWorkers?: number): Output
=> {
  const maxWorkersCount =
      numberOfWorkers ?? Math.floor(Object.keys(output.result).length);

  const timeCounts: Record<WorkerID, number> = {};
```

```
  const workersSorted: [WorkerID, Task[]][] = Object.entries(
      output.result,
  ).sort((a, b) => {
      const aTime = timeCounts[a[0]] ?? getDuration(a[1]);
      const bTime = timeCounts[b[0]] ?? getDuration(b[1]);
      timeCounts[a[0]] = aTime;
      timeCounts[b[0]] = bTime;

      return aTime - bTime;
  });

  const averageTime =
      workersSorted.reduce(
      (acc, [workerId, tasks]) => acc + timeCounts[workerId],
      0,
      ) / workersSorted.length;

  let leastTimeWorkerIndex = 0;
  for (
      let i = workersSorted.length - 1;
      i > workersSorted.length - 1 - maxWorkersCount;
      i--
  ) {
      if (leastTimeWorkerIndex >= workersSorted.length - 1) {
      break;
      }
      const [maxWorkerId, maxWorkerTasks] = workersSorted[i];
      const [minWorkerId, minWorkerTasks] =
workersSorted[leastTimeWorkerIndex];

      if (timeCounts[maxWorkerId] <= averageTime) {
      continue;
      }

      for (let j = 0; j < maxWorkerTasks.length; j++) {
      if (timeCounts[minWorkerId] >= averageTime) {
      leastTimeWorkerIndex += 1;
      i += 1;
      break;
      }
      const task = maxWorkerTasks[j];
      if (task.canBeDoneBy.includes(minWorkerId)) {
      minWorkerTasks.push(task);
      maxWorkerTasks.splice(j, 1);
      j -= 1;
      timeCounts[maxWorkerId] -= task.duration;
      timeCounts[minWorkerId] += task.duration;
      }
      }
  }

  return output;
};
```

## lib/utils/duration.ts

```
import { Task } from '../../types';

export const getDuration = (tasks: Task[]): number => {
```

```
  return tasks.reduce((acc, task) => acc + task.duration, 0);
};

export const getMaxDuration = (
  output: Record<string, Task[]>,
): number => {
  const durations = Object.values(output).map(getDuration);
  return Math.max(...durations);
};
```

## lib/utils/visualisation.ts

```
import { Output, Task } from '../../types';

export const visualize = (output: Output): string => {
  const tasksRows = Object.values(output.result);
  const result = tasksRows.map(visualizeRow).join('\n');
  return result;
};

const visualizeRow = (tasks: Task[]): string => {
  const tasksString = tasks.map(visualizeTask).join('--')
  return `W -> ${tasksString}`;
};

const visualizeTask = (task: Task): string => {
  return `${`▇`.repeat(task.duration)}`;
};
```

## lib/__tests__/method.test.ts

```
import { Input, Output, Task } from '../../types';
import { method } from '../method';
import { getDuration } from '../utils/duration';

describe('method', () => {
  it('should return an empty object if no workers or tasks are provided',
() => {
      const input: Input = {
      workers: [],
      tasks: [],
      };
      const expectedOutput: Output = {
      result: {},
      };
      const actualOutput = method(input);
      expect(actualOutput).toEqual(expectedOutput);
  });

  it('should assign all tasks to a single worker if there is only one
worker', () => {
      const input: Input = {
      workers: [{ id: '1', name: 'Worker 1' }],
      tasks: [
      {
            id: '1',
            name: 'Task 1',
```

```
            duration: 1,
            canBeDoneBy: ['1'],
            priority: 1,
        },
        {
            id: '2',
            name: 'Task 2',
            duration: 2,
            canBeDoneBy: ['1'],
            priority: 1,
        },
        ],
        };
        const expectedOutput: Output = {
        result: {
        '1': [
            {
            id: '1',
            name: 'Task 1',
            duration: 1,
            canBeDoneBy: ['1'],
            priority: 1,
            },
            {
            id: '2',
            name: 'Task 2',
            duration: 2,
            canBeDoneBy: ['1'],
            priority: 1,
            },
        ],
        },
        };
        const actualOutput = method(input);
        expect(actualOutput).toEqual(expectedOutput);
    });

    it('should assign tasks to workers based on their min time', () => {
        const input: Input = {
        workers: [
        { id: '1', name: 'Worker 1' },
        { id: '2', name: 'Worker 2' },
        ],
        tasks: [
        {
            id: '2',
            name: 'Task 2',
            duration: 2,
            canBeDoneBy: ['1', '2'],
            priority: 3,
        },
        {
            id: '1',
            name: 'Task 1',
            duration: 1,
            canBeDoneBy: ['1', '2'],
            priority: 4,
        },
        {
            id: '3',
            name: 'Task 3',
```

```
            duration: 3,
            canBeDoneBy: ['1', '2'],
            priority: 2,
      },
      {
            id: '4',
            name: 'Task 4',
            duration: 4,
            canBeDoneBy: ['1', '2'],
            priority: 1,
      },
      ],
      };

      const output = method(input);
      expect(getDuration(output.result['1'])).toEqual(5);
      expect(getDuration(output.result['2'])).toEqual(5);
  });
});
```

## lib/utils/visualisation.ts

```
import { Output, Task } from '../../types';

export const visualize = (output: Output): string => {
  const tasksRows = Object.values(output.result);
  const result = tasksRows.map(visualizeRow).join('\n');
  return result;
};

const visualizeRow = (tasks: Task[]): string => {
  const tasksString = tasks.map(visualizeTask).join('--')
  return `W -> ${tasksString}`;
};

const visualizeTask = (task: Task): string => {
  return `${`🏢`.repeat(task.duration)}`;
};
```

## lib/balance/__tests__/balance.test.ts

```
import { Output } from '../../../types';
import { balance } from '../balance';

const genTask = (workerId: string, taskId: string, canBeDoneBy: string[])
=> ({
  id: taskId,
  name: taskId,
  priority: 1,
  canBeDoneBy,
  duration: 1,
});

describe('balance', () => {
  it.each<{ before: Output; after: Output; workersToBalance?: number }>([
      {
      before: {
```

```
      result: {
            w1: [genTask('w1', 't1', ['w1'])],
            w2: [genTask('w2', 't2', ['w2']), genTask('w2', 't3',
['w2'])],
            w3: [
            genTask('w3', 't6', ['w1', 'w2', 'w3']),
            genTask('w3', 't7', ['w1', 'w2', 'w3']),
            genTask('w3', 't8', ['w1', 'w2', 'w3']),
            ],
      },
      },
      after: {
      result: {
            w1: [
            genTask('w1', 't1', ['w1']),
            genTask('w3', 't6', ['w1', 'w2', 'w3']),
            ],
            w2: [genTask('w2', 't2', ['w2']), genTask('w2', 't3',
['w2'])],
            w3: [
            genTask('w3', 't7', ['w1', 'w2', 'w3']),
            genTask('w3', 't8', ['w1', 'w2', 'w3']),
            ],
      },
      },
      },
      {
      before: {
      result: {
            w1: [genTask('w1', 't1', ['w1'])],
            w2: [
            genTask('w2', 't2', ['w1', 'w2']),
            genTask('w2', 't3', ['w1', 'w2']),
            genTask('w2', 't4', ['w2']),
            genTask('w2', 't5', ['w2']),
            ],
            w3: [
            genTask('w3', 't6', ['w1', 'w2', 'w3']),
            genTask('w3', 't7', ['w1', 'w2', 'w3']),
            genTask('w3', 't8', ['w1', 'w2', 'w3']),
            genTask('w3', 't9', ['w1', 'w2', 'w3']),
            genTask('w3', 't10', ['w1', 'w2', 'w3']),
            ],
      },
      },
      after: {
      result: {
            w1: [
            genTask('w1', 't1', ['w1']),
            genTask('w3', 't6', ['w1', 'w2', 'w3']),
            genTask('w3', 't7', ['w1', 'w2', 'w3']),
            ],
            w2: [
            genTask('w2', 't2', ['w1', 'w2']),
            genTask('w2', 't3', ['w1', 'w2']),
            genTask('w2', 't4', ['w2']),
            genTask('w2', 't5', ['w2']),
            ],
            w3: [
            genTask('w3', 't8', ['w1', 'w2', 'w3']),
            genTask('w3', 't9', ['w1', 'w2', 'w3']),
```

```
            genTask('w3', 't10', ['w1', 'w2', 'w3']),
            ],
      },
      },
      workersToBalance: 2,
      },
  ])('should balance the output', ({ before, after, workersToBalance })
=> {
      expect(balance(before, workersToBalance)).toEqual(after);
  });
});
```

## lib/vogelsApproximation/__tests__/mapTasksInputToTransportationProblemInput.test.ts

```
import { Input } from '../../../types';
import { mapTasksInputToTransportationProblemInput } from '../mappers';
import { TransportationProblemInput } from '../types';

describe('mapTasksInputToTransportationProblemInput', () => {
  it.each<{ input: Input; output: TransportationProblemInput }>([
      {
      input: {
      workers: [
            { id: '1', name: 'Worker 1' },
            {
            id: '2',
            name: 'Worker 2',
            },
            {
            id: '3',
            name: 'Worker 3',
            },
      ],
      tasks: [
            {
            id: '1',
            name: 'Task 1',
            duration: 1,
            canBeDoneBy: ['1', '2'],
            priority: 2,
            },
            {
            id: '2',
            name: 'Task 2',
            duration: 2,
            canBeDoneBy: ['1', '2'],
            priority: 3,
            },
            {
            id: '3',
            name: 'Task 3',
            duration: 3,
            canBeDoneBy: ['2'],
            priority: 4,
            },
            {
```

```
            id: '4',
            name: 'Task 4',
            duration: 4,
            canBeDoneBy: ['3'],
            priority: 5,
            },
      ],
      },
      output: {
      costs: [
            [1, 2, Number.MAX_SAFE_INTEGER, Number.MAX_SAFE_INTEGER],
            [1, 2, 3, Number.MAX_SAFE_INTEGER],
            [
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            4,
            ],
      ],
      demands: [
            expect.objectContaining({ id: '1', name: 'Task 1', demand: 1
}),
            expect.objectContaining({ id: '2', name: 'Task 2', demand: 2
}),
            expect.objectContaining({ id: '3', name: 'Task 3', demand: 3
}),
            expect.objectContaining({ id: '4', name: 'Task 4', demand: 4
}),
      ],
      suppliers: [
            { id: '1', name: 'Worker 1', supply: Number.MAX_SAFE_INTEGER
},
            { id: '2', name: 'Worker 2', supply: Number.MAX_SAFE_INTEGER
},
            { id: '3', name: 'Worker 3', supply: Number.MAX_SAFE_INTEGER
},
      ],
      },
      },
  ])(
      'should map tasks input to transportation problem input correctly',
      ({ input, output }) => {
      const result = mapTasksInputToTransportationProblemInput(input);

      expect(result).toEqual(output);
      },
  );
});
```

## lib/vogelsApproximation/__tests__/mapTransportationProblemOutputToTasksOutput.test.ts

```
import { Output, Task, Worker } from '../../../types';
import { mapTransportationProblemOutputToTasksOutput } from '../mappers';
import { TransportationProblemOutput } from '../types';

describe('mapTransportationProblemOutputToTasksOutput', () => {
```

```
  it.each<{ input: TransportationProblemOutput<Worker, Task>; output:
Output }>(
      [
      {
      input: {
            allocations: [
            {
            supplier: { id: '1', name: 'Worker 1' },
            destination: {
                  id: '1',
                  name: 'Task 1',
                  duration: 1,
                  canBeDoneBy: ['1', '2'],
                  priority: 2,
            },
            allocatedAmount: 1,
            },
            {
            supplier: { id: '1', name: 'Worker 1' },
            destination: {
                  id: '2',
                  name: 'Task 2',
                  duration: 2,
                  canBeDoneBy: ['1', '2'],
                  priority: 3,
            },
            allocatedAmount: 1,
            },
            {
            supplier: { id: '2', name: 'Worker 2' },
            destination: {
                  id: '3',
                  name: 'Task 3',
                  duration: 3,
                  canBeDoneBy: ['2'],
                  priority: 4,
            },
            allocatedAmount: 1,
            },
            ],
      },
      output: {
            result: {
            '1': [
            {
                  id: '1',
                  name: 'Task 1',
                  duration: 1,
                  canBeDoneBy: ['1', '2'],
                  priority: 2,
            },
            {
                  id: '2',
                  name: 'Task 2',
                  duration: 2,
                  canBeDoneBy: ['1', '2'],
                  priority: 3,
            },
            ],
            '2': [
            {
```

```
                  id: '3',
                  name: 'Task 3',
                  duration: 3,
                  canBeDoneBy: ['2'],
                  priority: 4,
            },
            ],
            },
      },
      },
      ],
  )(
      'should map transportation problem output to tasks output
correctly',
      ({ input, output }) => {
      const result = mapTransportationProblemOutputToTasksOutput(input);

      expect(result).toEqual(output);
      },
  );
});
```

## lib/vogelsApproximation/__tests__/vogelsApproximation.test.ts

```
import { Task, Worker } from '../../../types';
import {
  TransportationProblemInput,
  TransportationProblemOutput,
} from '../types';
import { vogelsApproximation } from '../vogelsApproximation';

describe('vogelsApproximation', () => {
  it.each<{
      input: TransportationProblemInput;
      output: TransportationProblemOutput;
  }>([
      {
      input: {
      costs: [
            [2, Number.MAX_SAFE_INTEGER, Number.MAX_SAFE_INTEGER, 5],
            [
            Number.MAX_SAFE_INTEGER,
            3,
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            ],
            [Number.MAX_SAFE_INTEGER, 3, 4, Number.MAX_SAFE_INTEGER],
      ],
      demands: [
            { id: '1', name: 'Task 1', demand: 2 },
            { id: '2', name: 'Task 2', demand: 3 },
            { id: '3', name: 'Task 3', demand: 4 },
            { id: '4', name: 'Task 4', demand: 5 },
      ],
      suppliers: [
            { id: '1', name: 'Worker 1', supply: Number.MAX_SAFE_INTEGER
},
```

```
            { id: '2', name: 'Worker 2', supply: Number.MAX_SAFE_INTEGER
},
            { id: '3', name: 'Worker 3', supply: Number.MAX_SAFE_INTEGER
},
      ],
      },
      output: {
      allocations: [
            {
            allocatedAmount: 2,
            destination: expect.objectContaining({
            id: '1',
            name: 'Task 1',
            }),
            supplier: expect.objectContaining({
            id: '1',
            name: 'Worker 1',
            }),
            },
            {
            allocatedAmount: 3,
            destination: expect.objectContaining({
            id: '2',
            name: 'Task 2',
            }),
            supplier: expect.objectContaining({
            id: '2',
            name: 'Worker 2',
            }),
            },
            {
            allocatedAmount: 4,
            destination: expect.objectContaining({
            id: '3',
            name: 'Task 3',
            }),
            supplier: expect.objectContaining({
            id: '3',
            name: 'Worker 3',
            }),
            },
            {
            allocatedAmount: 5,
            destination: expect.objectContaining({
            id: '4',
            name: 'Task 4',
            }),
            supplier: expect.objectContaining({
            id: '1',
            name: 'Worker 1',
            }),
            },
      ],
      },
      },
      {
      input: {
      costs: [
            [2, 3, 4, 5],
            [
            Number.MAX_SAFE_INTEGER,
```

```
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            ],
            [
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            ],
      ],
      demands: [
            { id: '1', name: 'Task 1', demand: 2 },
            { id: '2', name: 'Task 2', demand: 3 },
            { id: '3', name: 'Task 3', demand: 4 },
            { id: '4', name: 'Task 4', demand: 5 },
      ],
      suppliers: [
            { id: '1', name: 'Worker 1', supply: Number.MAX_SAFE_INTEGER
},
            { id: '2', name: 'Worker 2', supply: Number.MAX_SAFE_INTEGER
},
            { id: '3', name: 'Worker 3', supply: Number.MAX_SAFE_INTEGER
},
      ],
      },
      output: {
      allocations: [
            {
            allocatedAmount: 2,
            destination: expect.objectContaining({
            id: '1',
            name: 'Task 1',
            }),
            supplier: expect.objectContaining({
            id: '1',
            name: 'Worker 1',
            }),
            },
            {
            allocatedAmount: 3,
            destination: expect.objectContaining({
            id: '2',
            name: 'Task 2',
            }),
            supplier: expect.objectContaining({
            id: '1',
            name: 'Worker 1',
            }),
            },
            {
            allocatedAmount: 4,
            destination: expect.objectContaining({
            id: '3',
            name: 'Task 3',
            }),
            supplier: expect.objectContaining({
            id: '1',
            name: 'Worker 1',
            }),
            },
```

```
            {
            allocatedAmount: 5,
            destination: expect.objectContaining({
            id: '4',
            name: 'Task 4',
            }),
            supplier: expect.objectContaining({
            id: '1',
            name: 'Worker 1',
            }),
            },
      ],
      },
      },
      {
      input: {
      costs: [
            [
            2,
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            ],
            [
            Number.MAX_SAFE_INTEGER,
            3,
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            ],
            [
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            4,
            Number.MAX_SAFE_INTEGER,
            ],
            [
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            Number.MAX_SAFE_INTEGER,
            5,
            ],
      ],
      demands: [
            { id: '1', name: 'Task 1', demand: 2 },
            { id: '2', name: 'Task 2', demand: 3 },
            { id: '3', name: 'Task 3', demand: 4 },
            { id: '4', name: 'Task 4', demand: 5 },
      ],
      suppliers: [
            { id: '1', name: 'Worker 1', supply: Number.MAX_SAFE_INTEGER
},
            { id: '2', name: 'Worker 2', supply: Number.MAX_SAFE_INTEGER
},
            { id: '3', name: 'Worker 3', supply: Number.MAX_SAFE_INTEGER
},
            { id: '4', name: 'Worker 4', supply: Number.MAX_SAFE_INTEGER
},
      ],
      },
      output: {
      allocations: [
```

```
      {
      allocatedAmount: 2,
      destination: expect.objectContaining({
      id: '1',
      name: 'Task 1',
      }),
      supplier: expect.objectContaining({
      id: '1',
      name: 'Worker 1',
      }),
      },
      {
      allocatedAmount: 3,
      destination: expect.objectContaining({
      id: '2',
      name: 'Task 2',
      }),
      supplier: expect.objectContaining({
      id: '2',
      name: 'Worker 2',
      }),
      },
      {
      allocatedAmount: 4,
      destination: expect.objectContaining({
      id: '3',
      name: 'Task 3',
      }),
      supplier: expect.objectContaining({
      id: '3',
      name: 'Worker 3',
      }),
      },
      {
      allocatedAmount: 5,
      destination: expect.objectContaining({
      id: '4',
      name: 'Task 4',
      }),
      supplier: expect.objectContaining({
      id: '4',
      name: 'Worker 4',
      }),
      },
],
},
},
{
input: {
costs: [
      [2, Number.MAX_SAFE_INTEGER, Number.MAX_SAFE_INTEGER, 5],
      [
      Number.MAX_SAFE_INTEGER,
      3,
      Number.MAX_SAFE_INTEGER,
      Number.MAX_SAFE_INTEGER,
      ],
      [2, 3, 4, 5],
      [Number.MAX_SAFE_INTEGER, Number.MAX_SAFE_INTEGER, 4, 5],
],
demands: [
```

```
            { id: '1', name: 'Task 1', demand: 2 },
            { id: '2', name: 'Task 2', demand: 3 },
            { id: '3', name: 'Task 3', demand: 4 },
            { id: '4', name: 'Task 4', demand: 5 },
      ],
      suppliers: [
            { id: '1', name: 'Worker 1', supply: Number.MAX_SAFE_INTEGER
},
            { id: '2', name: 'Worker 2', supply: Number.MAX_SAFE_INTEGER
},
            { id: '3', name: 'Worker 3', supply: Number.MAX_SAFE_INTEGER
},
            { id: '4', name: 'Worker 4', supply: Number.MAX_SAFE_INTEGER
},
      ],
      },
      output: {
      allocations: [
            {
            allocatedAmount: 3,
            destination: expect.objectContaining({
            id: '2',
            name: 'Task 2',
            }),
            supplier: expect.objectContaining({
            id: '2',
            name: 'Worker 2',
            }),
            },
            {
            allocatedAmount: 2,
            destination: expect.objectContaining({
            id: '1',
            name: 'Task 1',
            }),
            supplier: expect.objectContaining({
            id: '1',
            name: 'Worker 1',
            }),
            },
            {
            allocatedAmount: 5,
            destination: expect.objectContaining({
            id: '4',
            name: 'Task 4',
            }),
            supplier: expect.objectContaining({
            id: '1',
            name: 'Worker 1',
            }),
            },
            {
            allocatedAmount: 4,
            destination: expect.objectContaining({
            id: '3',
            name: 'Task 3',
            }),
            supplier: expect.objectContaining({
            id: '3',
            name: 'Worker 3',
            }),
```

```
            },
        ],
        },
        },
        {
        input: {
        costs: [
              [2, 3, 4, 5],
              [2, 3, Number.MAX_SAFE_INTEGER, 5],
              [Number.MAX_SAFE_INTEGER, 3, 4, 5],
              [2, Number.MAX_SAFE_INTEGER, 4, Number.MAX_SAFE_INTEGER],
        ],
        demands: [
              { id: '1', name: 'Task 1', demand: 2 },
              { id: '2', name: 'Task 2', demand: 3 },
              { id: '3', name: 'Task 3', demand: 4 },
              { id: '4', name: 'Task 4', demand: 5 },
        ],
        suppliers: [
              { id: '1', name: 'Worker 1', supply: Number.MAX_SAFE_INTEGER
},
              { id: '2', name: 'Worker 2', supply: Number.MAX_SAFE_INTEGER
},
              { id: '3', name: 'Worker 3', supply: Number.MAX_SAFE_INTEGER
},
              { id: '4', name: 'Worker 4', supply: Number.MAX_SAFE_INTEGER
},
        ],
        },
        output: {
        allocations: [
              {
              allocatedAmount: 2,
              destination: expect.objectContaining({
              id: '1',
              name: 'Task 1',
              }),
              supplier: expect.objectContaining({
              id: '4',
              name: 'Worker 4',
              }),
              },
              {
              allocatedAmount: 4,
              destination: expect.objectContaining({
              id: '3',
              name: 'Task 3',
              }),
              supplier: expect.objectContaining({
              id: '4',
              name: 'Worker 4',
              }),
              },
              {
              allocatedAmount: 3,
              destination: expect.objectContaining({
              id: '2',
              name: 'Task 2',
              }),
              supplier: expect.objectContaining({
              id: '1',
```

```
            name: 'Worker 1',
            }),
            },
            {
            allocatedAmount: 5,
            destination: expect.objectContaining({
            id: '4',
            name: 'Task 4',
            }),
            supplier: expect.objectContaining({
            id: '1',
            name: 'Worker 1',
            }),
            },
      ],
      },
      },
  ])('should return correct output', ({ input, output }) => {
      expect(
      vogelsApproximation(input as TransportationProblemInput<Worker,
Task>),
      ).toEqual(output);
  });
});
```

**Додаток 3**

**Копія презентації**

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

# Спосіб та програмне забезпечення для організації та мінімізації часу виконання задач

Доповідач: Коваль Андрій Олександрович
Науковий керівник: к.т.н., старший викладач Хіцко Яна Володимирівна

Київ – 2023

# АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ

- Підвищення продуктивності та ефективності робочих процесів
- Оптимізація розподілу ресурсів та робочої сили

# НАУКОВЕ ЗАВДАННЯ ТА МЕТА

**Наукове завдання:** вдосконалити та модифікувати існуючі алгоритми розподілу задач.

**Мета дослідження**: розробка та валідація алгоритму, який зможе динамічно обирати підхід для оптимізації порядку виконання задач залежно від розміру набору даних.

**Об’єкт дослідження**: процес планування порядку виконання задач та їх розподілення між працівниками з мінімізацією часу простою.

**Предмет дослідження:** методи, алгоритми та програмні засоби для оптимізації процесу планування та розподілу задач.

## Окремі завдання

1. Огляд існуючих рішень
2. Модифікація існуючих методів
3. Розробка власного методу
4. Аналіз отриманих результатів

# ОСНОВНІ ПРОГРАМНІ СИСТЕМИ ДЛЯ ВПОРЯДКУВАННЯ ЗАДАЧ


OptaPlanner

Google OR-Tools

IBM CPLEX

# АНАЛІЗ ІСНУЮЧИХ СИСТЕМ


## Недоліки існуючих системи

- Складність навчання
- Потреба у специфічній технічній експертизі

## Переваги запропонованої системи

- Простіша альтернатива
- Вирішення однією конкретної задачі

# ПОТЕНЦІЙНІ АЛГОРИТМИ ДЛЯ ВИРІШЕННЯ НАШОЇ ЗАДАЧІ

1. Алгоритми перебору
2. Алгоритм мінімального кістякового дерева
3. Генетичні алгоритми
4. Алгоритми пошуку шляху
5. Алгоритми для вирішення задачі транспортування

**Недоліки існуючих алгоритмів**

- Час виконання (алгоритм перебору)
- Евристичність, неможливість отримання достеменно точного результату
- Недостатня відповідність задачі, більша сфокусованість на інших сферах
- Недостатня гнучкість деяких алгоритмів

**Переваги алгоритму для вирішення задачі транспортування**

- Часткова схожість з задачею розподілення задач між робітниками

# ГІПОТЕЗИ

1. Одним із основних алгоритмів роботи може бути Алгоритм апроксимації Фогеля
2. Результуючий алгоритм можна розбити на 3 окремі частини: повний перебір, апроксимація Фогеля, покращення результатів апроксимації
3. Покращення результатів апроксимації може бути реалізоване у вигляді евристичного алгоритму балансування результуючого розподілення задач

# КОМБІНОВАНИЙ АЛГОРИТМ ВИРІШЕННЯ ЗАДАЧІ РОЗПОДІЛЕННЯ ЗАДАЧ


1. Повний перебір
2. Апроксимація Фогеля
3. Балансування результатів апроксимації

# КОМБІНОВАНИЙ АЛГОРИТМ ВИРІШЕННЯ ЗАДАЧІ РОЗПОДІЛЕННЯ ЗАДАЧ


**Повний перебір**

- Корисний на малій кількості даних (<50~ елементів)
- Має факторіальну складність O(N!)

Етапи алгоритму:

- Генерація всіх можливих комбінацій розподілу задач
- Вибір найкращої комбінації

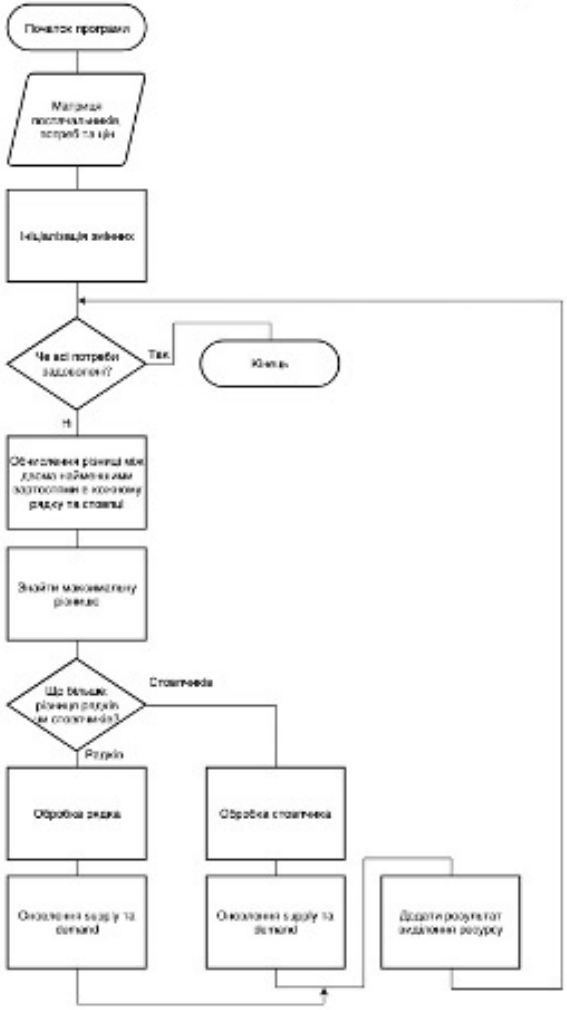
# КОМБІНОВАНИЙ АЛГОРИТМ ВИРІШЕННЯ ЗАДАЧІ РОЗПОДІЛЕННЯ ЗАДАЧ


**Алгоритм апроксимації Фогеля**

Етапи алгоритму:

- Обчислення штрафів
- Вибір осередку з максимальним штрафом.
- Коригування поставок та попиту
- Ітерація та завершення. Процес повторюється, поки всі поставки та попити не будуть задоволені

# АЛГОРИТМ АПРОКСИМАЦІЇ ФОГЕЛЯ



# КОМБІНОВАНИЙ АЛГОРИТМ ВИРІШЕННЯ ЗАДАЧІ РОЗПОДІЛЕННЯ ЗАДАЧ


**Алгоритм апроксимації Фогеля**

- Часова складність алгоритму O(k(nm log(nm)), де k – кількість ітерацій головного циклу, а m і n кількість рядків і стовпців

# КОМБІНОВАНИЙ АЛГОРИТМ ВИРІШЕННЯ ЗАДАЧІ РОЗПОДІЛЕННЯ ЗАДАЧ

## Алгоритм апроксимації Фогеля

| | d1 | d2 | d3 | d4 | Постачання |
|---|---|---|---|---|---|
| s1 | 3 | 2 | 7 | 6 | 50 |
| s2 | 7 | 5 | 2 | 3 | 60 |
| s3 | 2 | 5 | 4 | 5 | 25 |
| Потреба | 60 | 40 | 20 | 15 | |

# КОМБІНОВАНИЙ АЛГОРИТМ ВИРІШЕННЯ ЗАДАЧІ РОЗПОДІЛЕННЯ ЗАДАЧ

## Алгоритм апроксимації Фогеля

| | d1 | d2 | d3 | d4 | Постачання |
|---|---|---|---|---|---|
| s1 | 3 (10) | 2 (40) | 7 | 6 | 50 |
| s2 | 7 (25) | 5 | 2 (20) | 3 (15) | 60 |
| s3 | 2 (25) | 5 | 4 | 5 | 25 |
| Потреба | 60 | 40 | 20 | 15 | |

# КОМБІНОВАНИЙ АЛГОРИТМ ВИРІШЕННЯ ЗАДАЧІ РОЗПОДІЛЕННЯ ЗАДАЧ


## Алгоритм апроксимації Фогеля

| | t1 | t2 | t3 | t4 | $t_{dummy}$ | Постачання |
|---|---|---|---|---|---|---|
| w1 | 2 | 3 | INF | INF | INF | INF |
| w2 | INF | INF | 4 | 5 | INF | INF |
| w3 | INF | 3 | 4 (4) | INF | INF | INF |
| Потреба часу | 2 | 3 | 4 | 5 | INF | |

# КОМБІНОВАНИЙ АЛГОРИТМ ВИРІШЕННЯ ЗАДАЧІ РОЗПОДІЛЕННЯ ЗАДАЧ

## Алгоритм апроксимації Фогеля

| | t1 | t2 | t3 | t4 | $t_{dummy}$ | Постачання |
|---|---|---|---|---|---|---|
| w1 | 2 (2) | 3 (3) | INF | INF | INF (INF) | INF |
| w2 | INF | INF | 4 | 5 (5) | INF (INF) | INF |
| w3 | INF | 3 | 4 (4) | INF | INF (INF) | INF |
| Потреба часу | 2 | 3 | 4 | 5 | INF | |

# КОМБІНОВАНИЙ АЛГОРИТМ ВИРІШЕННЯ ЗАДАЧІ РОЗПОДІЛЕННЯ ЗАДАЧ


**Балансування результатів апроксимації**

Етапи алгоритму:

- Визначення кількості виконавців для балансування
- Розрахунок сумарного часу роботи для кожного виконавця
- Сортування виконавців за часом виконання задач
- Перерозподіл задач між виконавцями
- Оновлення часу виконання задач

# КОМБІНОВАНИЙ АЛГОРИТМ ВИРІШЕННЯ ЗАДАЧІ РОЗПОДІЛЕННЯ ЗАДАЧ


**Балансування результатів апроксимації**

Часова складність $O(Wlog2(W) + W_mT)$, де $W$ - кількість виконавців, $W_m$ - кількість виконавців, які балансуються, $T$- сумарна кількість задач

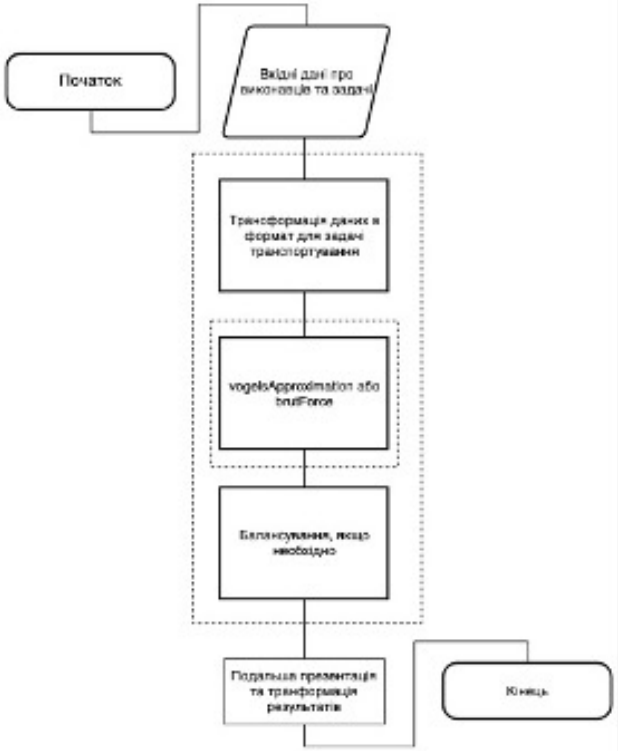
# ПРОГРАМНА РЕАЛІЗАЦІЯ



# ПРОГРАМНА РЕАЛІЗАЦІЯ

Програмна реалізація складається з таких етапів:

1. Генерація вхідних даних
2. Робота вибраного алгоритму в залежності від розміру вхідних даних
3. Аналіз отриманих результатів.



# ПРОГРАМНА РЕАЛІЗАЦІЯ

Наочні результати виконання алгоритмів на декількох наборах даних.

## Повний перебір

| Задачі | Виконавці | ОР (Одиниця роботи, Задачі * Виконавці) | Час (мс) |
|---|---|---|---|
| 5 | 3 | 15 | 0,13 |
| 10 | 3 | 30 | 0,76 |
| 10 | 5 | 50 | 12,9 |

# ПРОГРАМНА РЕАЛІЗАЦІЯ

Наочні результати виконання алгоритмів на декількох наборах даних.

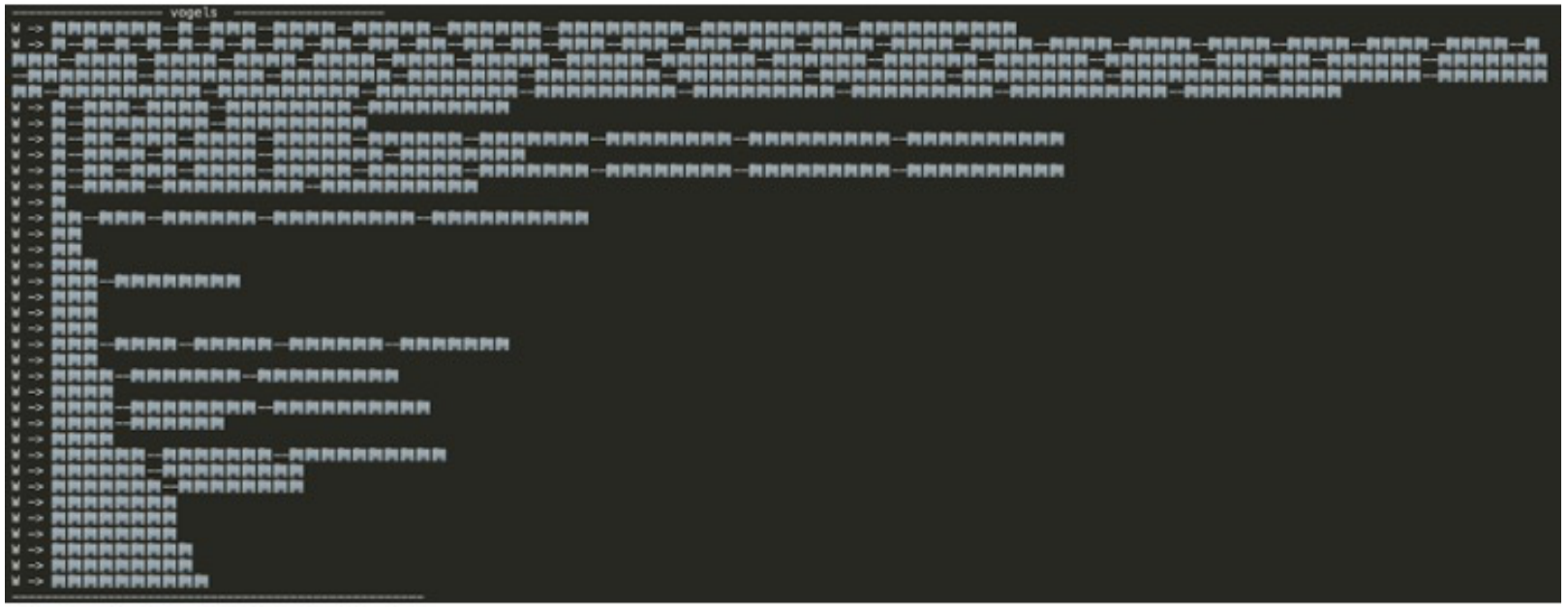
**Апроксимація Фогеля:**

| Задачі | Виконавці | ОР (Одиниця роботи, Задачі * Виконавці) | Час (мс) |
|---|---|---|---|
| 10 | 3 | 30 | 0,26 |
| 10 | 5 | 50 | 0,1 |
| 20 | 10 | 200 | 0,52 |
| 30 | 20 | 600 | 2,6 |
| 100 | 30 | 3000 | 32,4 |
| 150 | 50 | 7500 | 116,9 |
| 160 | 50 | 8000 | 140,8 |

Результати з балансуванням після апроксимації Фогеля відрізняються від результатів без балансування на +- 2-3%

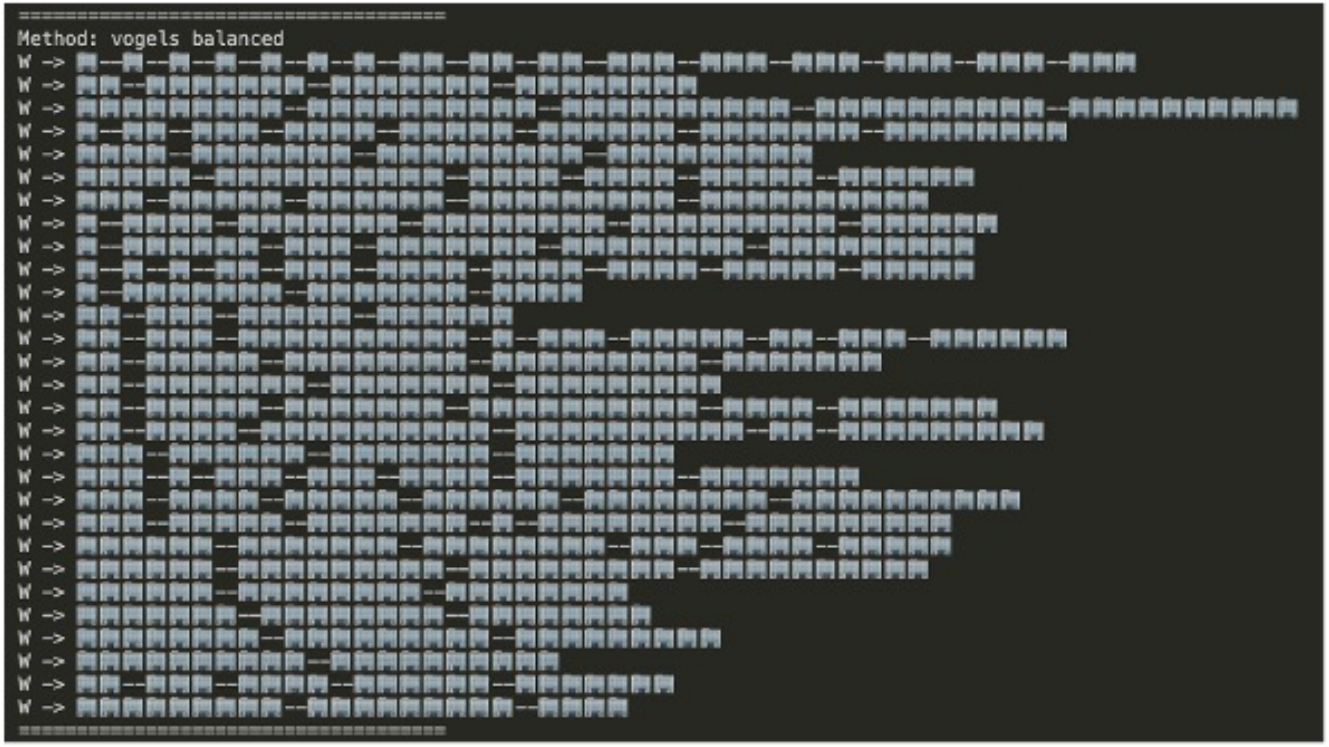
# ПРОГРАМНА РЕАЛІЗАЦІЯ

Приклад роботи без балансування



# ПРОГРАМНА РЕАЛІЗАЦІЯ

Приклад роботи з балансуванням



# ВИСНОВКИ

- Проаналізовано існуючі методи та сервіси для аналізу порядку виконання задач.
- Розроблено комбінований алгоритм з трьох частин і програмне рішення для оптимізації процесу розподілу завдань.
- Проаналізовано функціональні і нефункціональні вимоги і створено консольний додаток на базі досконалого алгоритму.
- Проведено аналіз результатів. Швидкодія є конкурентоспроможною серед аналогів.
- Розроблено бізнес-модель із гнучкими підходами до монетизації.

# ПЕРЕВІРКА НА ПЛАГІАТ

UNICHECK
by Turnitin

User name:
**Хіцко Яна Володимирівна**

Check ID:
**1016047768**

Check date:
**07.01.2024 21:42:21 EET**

Check type:
**Doc vs Internet + Library**

Report date:
**07.01.2024 21:42:48 EET**

User ID:
**91600**

File name: **ДИПЛОМ Коваль записка – копія для плагіату**

Page count: **63** Word count: **12467** Character count: **101755** File size: **273.97 KB** File ID: **1015747403**

## 4.33% Matches

Highest match: **0.77%** with Library source (File ID: **1015747404**)

| | | |
|---|---|---|
| 2.83% Internet sources | 375 | Page 65 |
| 3.28% Library sources | 425 | Page 68 |

## 0.27% Quotes

| | | |
|---|---|---|
| Quotes | 1 | Page 69 |

Exclusion of references is off

## 0% Exclusions

No exclusions

# АПРОБАЦІЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

XVI наукова конференція магістрантів та аспірантів
«Прикладна математика та комп'ютинг» (ПМК-2023)


***Дякую за увагу!***