АНОТАЦІЯ

*Зм.*

*Арк.*

*Прізвище*

*Підпис*

*Дата*

*Арк.*

2

ДП ІС-3235.1393-с.ПЗ

*Розроб.*

*Сінюков Д.Г.*

*Перевірив.*

*.*

*Павлов О.А.*

*Н. кон.*

*Жураковська О.С.*

*Затв.*

*Муха І.П.*

*Система підтримки розробки та аналізу ПДС-алгоритмів побудови розкладів робіт на паралельних верстатах однакової продуктивності*

*Літ.*

*Аркушів*

67

–

*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ*

*кафедра АСОІУ гр. ІС-32*

**Структура та обсяг роботи.** Пояснювальна записка дипломного проекту складається з п’яти розділів, містить 16 рисунків, 12 таблиць, 3 додатків, 8 джерел.

Дипломний проект присвячений розробці системи підтримки розробки та аналізу ПДС-алгоритмів побудови розкладів робіт на паралельних верстатах однакової продуктивності з метою підвищення ефективності процесу командної розробки ПДС-алгоритмів.

В дипломному проекті були розглянуті: метод гілок та меж і ПДС-алгоритми розв’язку задач побудови розкладів робіт на паралельних верстатах однакової продуктивності.

У розділі з інформаційного забезпечення були визначені вхідні та вихідні дані системи, було описано формат вхідних файлів і наведено схему бази даних системи.

У розділі з математичного забезпечення було описано векторний критерій побудови розкладів робіт на паралельних верстатах однакової продуктивності, обґрунтовано створений ПДС-алгоритм, порівняно його ефективність з методом гілок та меж точного розв’язку задачі і проведено експериментальні дослідження ефективності розробленого алгоритму.

У розділі програмного забезпечення описані основні засоби розробки системи, висунуті вимоги до технічного забезпечення, обрано та обґрунтовано архітектуру програмного забезпечення.

У технологічному розділі описана інструкція користувача.

|  |
| --- |
| ПДС-АЛГОРИТМ, РОЗКЛАД, МЕТОД ГІЛОК ТА МЕЖ, ЗАВДАННЯ, ПАРАЛЕЛЬНІ ВЕРСТАТИ, КАЛЕНДАРНЕ ПЛАНУВАННЯ, ВЕКТОРНИЙ КРИТЕРІЙ, ДИРЕКТИВНИЙ ТЕРМІН |
|  |

ABSTRACT

**Structure and content of the work**. The explanatory note to the diploma project consists of five chapters, contains 16 figures, 12 tables, 3 annexes, 8 references.

The diploma project is dedicated to development of the system of development support and analysis of PDS-algorithms for building task schedules on parallel machines with same performances with the purpose of increasing the efficiency of the PDS-algorithms development process.

Branch and bound method and PDS-algorithms for building task schedules on parallel machines with same performances were considered in this diploma project.

In the section of information support input and output data was determined, there was described the format of the data, and specified the database scheme.

The section of mathematical support described vector criterion of building task schedules on parallel machines with same performances, we substantiated the new PDS-algorithm, its efficiency was compared with the precise method, and the experimental research of the algorithms’ efficiency was carried out.

In the section of software we described basic software development tools, put forward technical requirements, chose and substantiated the software architecture.

The user’s manual was provided in the technological section.

|  |
| --- |
| PDS-ALGORITHM, SCHEDULE, BRANCH AND BOUND, TASK, PARALLEL MACHINES, SCHEDULING, VECTOR CRITERION, DEADLINE |
|  |

ЗМІСТ

[Вступ 6](#_Toc483987318)

[1 Загальні положення 7](#_Toc483987319)

[**1.1** **Опис предметного середовища** 7](#_Toc483987320)

[**1.1.1** **Опис процесу діяльності** 8](#_Toc483987321)

[**1.1.2** **Опис функціональної моделі** 9](#_Toc483987322)

[**1.2** **Огляд наявних аналогів** 11](#_Toc483987323)

[**1.3** **Постановка задачі** 13](#_Toc483987324)

[**1.3.1** **Призначення розробки** 13](#_Toc483987325)

[**1.3.2** **Цілі та задачі розробки** 13](#_Toc483987326)

[**Висновок до розділу** 14](#_Toc483987327)

[2 Інформаційне забезпечення 15](#_Toc483987328)

[**2.1** **Вхідні дані** 15](#_Toc483987329)

[**2.1.1** **Аналітики** 15](#_Toc483987330)

[**2.1.2** **Набори вхідних даних аналітик** 15](#_Toc483987331)

[**2.1.3** **Алгоритми** 16](#_Toc483987332)

[**2.1.4** **Вхідні дані для побудови розкладу** 16](#_Toc483987333)

[**2.2** **Вихідні дані** 18](#_Toc483987334)

[**2.3** **Опис структури бази даних** 19](#_Toc483987335)

[**Висновок до розділу** 21](#_Toc483987336)

[3 Математичне забезпечення 22](#_Toc483987337)

[**3.1** **Змістовна постановка задачі** 22](#_Toc483987338)

[**3.2** **Математична постановка задачі** 22](#_Toc483987339)

[**3.3** **Обґрунтування методу розв’язання** 23](#_Toc483987340)

[**3.4** **Опис методів розв’язання** 24](#_Toc483987341)

[**3.4.1** **Опис ПДС-алгоритму розв’язання задачі** 24](#_Toc483987342)

[**3.4.2** **Експериментальне дослідження обчислювальної складності алгоритмів** 26](#_Toc483987343)

[**Висновок до розділу** 27](#_Toc483987344)

[4 Програмне та технічне забезпечення 29](#_Toc483987345)

[**4.1** **Засоби розробки** 29](#_Toc483987346)

[**4.2** **Вимоги до технічного забезпечення** 30](#_Toc483987347)

[**4.2.1** **Загальні вимоги** 30](#_Toc483987348)

[**4.3** **Архітектура програмного забезпечення** 30](#_Toc483987349)

[**4.3.1** **Діаграма класів** 30](#_Toc483987350)

[**4.3.2** **Діаграма послідовності** 32](#_Toc483987351)

[**4.3.3** **Діаграма компонентів** 32](#_Toc483987352)

[**4.3.4** **Специфікація функцій** 33](#_Toc483987353)

[**Висновок до розділу** 34](#_Toc483987354)

[5 Технологічний розділ 35](#_Toc483987355)

[**5.1** **Керівництво користувача** 35](#_Toc483987356)

[**5.2** **Випробування програмного продукту** 43](#_Toc483987357)

[**5.2.1** **Мета випробувань** 44](#_Toc483987358)

[**5.2.2** **Загальні положення** 44](#_Toc483987359)

[**5.2.3** **Результати випробувань** 44](#_Toc483987360)

[**Висновок до розділу** 46](#_Toc483987361)

[ЗАГАЛЬНІ ВИСНОВКИ 48](#_Toc483987362)

[Перелік посилань 50](#_Toc483987363)

Вступ

На сьогоднішній день однією з відомих і важливих областей прикладної математики є теорія розкладів. Висока популярність даної області зумовлена, в першу чергу, існуванням великої кількості прикладних задач, які потребують ефективних методів розв’язання. В таких предметних областях як: виробництво, управління персоналом, управління проектами, транспорт, обчислювальні системи, методи розв’язання задач складання розкладів і календарного планування посідають ключове місце.

Втім, сьогодні існує багато задач, які не мають точних методів розв’язання, або точний метод існує, але він не є ефективним (його обчислювальна складність є занадто високою при реальних розмірностях вхідних даних). Для ефективного розв’язання таких задач було винайдено новий клас алгоритмів – ПДС-алгоритми (детальніше ПДС-алгоритми будуть описані в підрозділі опису предметного середовища 1-го розділу).

Беручи до уваги складність побудови алгоритмів вирішення задач складання розкладів а також те, що сьогодні є високий попит в ефективному вирішенні таких задач, існує значна потреба в інформаційній системі, яка б допомагала науковцям у розробці ПДС-алгоритмів.

Даний дипломний проект присвячений розробці системи підтримки розробки та аналізу ПДС-алгоритмів побудови розкладів робіт на паралельних верстатах однакової продуктивності.

**Практичне значення одержаних результатів.** Розроблено точний алгоритм побудови оптимальних за критерієм максимізації мінімального моменту початку роботи верстатів розкладів робіт з довільними директивними термінами на паралельних верстатах однакової продуктивності.

# Загальні положення

## **Опис предметного середовища**

Об’єктом автоматизації є задачі побудови розкладів робіт з довільними директивними термінами на паралельних верстатах однакової продуктивності.

**ПДС-алгоритмом** називається алгоритм, що складається з поліноміальної складової і експоненціального підалгоритму чи поліноміальної апроксимації точного алгоритму, який може включати умови декомпозиції початкової задачі на підзадачі меншої розмірності [1].

Поліноміальна складова породжується логіко-аналітичними умовами (p-умовами), виконання яких допустимим рішенням, отриманим в результаті реалізації поліноміальної складової ПДС-алгоритму, визначає його як оптимальне. p-умови знаходяться в результаті теоретичних досліджень відповідного класу важковирішуваних задач комбінаторної оптимізації. Поліноміальна складова ПДС-алгоритму синтезується таким чином, щоб послідовна процедура конструювання допустимих рішень була найбільш ефективною з точки зору реалізації p-умов (достатніх ознак оптимальності допустимих рішень). Верхня оцінка складності поліноміальної складової відома. Іноді експоненціальна складова ПДС-алгоритму замінюється алгоритмом поліноміальної складності, що призводить до наближеного (субоптимального) рішення [1].

В [2] формулюється ряд нових одноетапних задач календарного планування, для кожної з яких знаходяться p-умови (ознаки оптимальності допустимого рішення). Отримані результати реалізують можливість побудови для цих класів комбінаторних задач оптимізації ефективних ПДС-алгоритмів.

Таким чином, розширюється клас задач, для яких можуть бути побудовані ПДС-алгоритми. Наведені в даному розділі задачі не досліджувалися на предмет того, до якого класу задач (P або не простіше, ніж NP-повні) вони відносяться. Однак це не є важливим по тій причині, що ПДС-алгоритми є ефективними і в наступних випадках:

* не знайдено точного поліноміального алгоритму вирішення задачі;
* точний поліноміальний алгоритм знайдений, але його обчислювальна складність є значно вищою за обчислювальну складність поліноміальної складової відповідного ПДС-алгоритму, при тому що поліноміальна складова ПДС-алгоритму є достатньо ефективною (реалізується достатньо часто для задоволення вимог конкретної предметної області).

Далі буде описано задачу складання оптимальних за векторним критерієм максимізації мінімального моменту початку роботи верстатів розкладів робіт з довільними директивними термінами на паралельних верстатах однакової продуктивності.

**Розклад** називається **допустимим**, якщо в ньому немає робіт із запізненням. **Робота** називається **допустимою**, якщо вона виконується без запізнення [3].

Допустимий розклад, у якого моменти запуску верстатів є найбільш пізніми з урахуванням векторного (лексикографічного) порядку, називається **оптимальним за векторним** (лексикографічним) **критерієм** максимізації мінімального моменту початку роботи верстатів [3].

### **Опис процесу діяльності**

Розглянемо дії, які виконує розробник під час основного процесу роботи в системі – тестування та аналізу нового ПДС-алгоритму.

Процес: «Тестування та аналіз алгоритму».

Передумови: користувач має бути авторизованим.

Послідовність дій розробника при тестуванні та аналізі алгоритму проілюстровано на UML діаграмі діяльності, яку наведено у частині графічного матеріалу дипломного проекту.

### **Опис функціональної моделі**

Акторами системи є:

* неавтентифікований користувач;
* розробник алгоритмів;
* адміністратор.

**Неавтентифікований користувач** має можливість створити обліковий запис та увійти в систему.

**Розробнику алгоритмів** доступні основні функції системи: ті, що стосуються його облікового запису, його алгоритмів, а також функції перегляду та аналізу публічних алгоритмів інших користувачів.

**Адміністратор** – актор, що має найбільші права і відповідальність у системі. Крім функцій розробника алгоритмів, адміністратор має безпосередній доступ до бази даних системи. Найважливіша функція адміністратора – ведення даних, що передаються на вхід алгоритмам користувачів при їх тестуванні та аналізі.

Визначимо дії, які можуть виконувати актори системи. Актори системи з їх діями зображені на схемі структурній варіантів використання у частині графічного матеріалу.

Таблиця 1.1 *–*­ Опис варіантів використання

|  |  |  |  |
| --- | --- | --- | --- |
| **Актор** | **Назва варіанту** | **Опис** | **Пріоритет** |
| Неавтентифікований користувач | Вхід у систему | Неавтентифікований користувач входить у систему | Високий |

Продовження таблиці 1.1

|  |  |  |  |
| --- | --- | --- | --- |
| **Актор** | **Назва варіанту** | **Опис** | **Пріоритет** |
| Неавтентифікований користувач | Створення облікового запису | Користувач без облікового запису створює обліковий запис у системі | Високий |
| Розробник алгоритмів | Ведення власних алгоритмів | Розробник алгоритмів може додавати, редагувати та видаляти власні алгоритми | Високий |
| Публікація алгоритму | Користувач може опублікувати власний алгоритм, після чого алгоритм стане доступним для перегляду іншим користувачам | Низький |
| Побудова розкладу для довільних вхідних даних | Користувач обирає алгоритм, який будує розклад для введених вхідних даних, діаграма Ґанта розкладу відображається на екрані | Середній |
| Перегляд публічних алгоритмів інших користувачів | Користувач бачить опубліковані алгоритми інших користувачів і може переглядати їх інформацію | Низький |
| Аналіз ефективності алгоритму | Розробник переглядає графічні представлення аналітик алгоритму, що дають йому інформацію про ефективність алгоритму і його «слабкі місця» | Високий |

Продовження таблиці 1.1

|  |  |  |  |
| --- | --- | --- | --- |
| **Актор** | **Назва варіанту** | **Опис** | **Пріоритет** |
| Розробник алгоритмів | Прогін алгоритму на наборах вхідних даних | Перед виведенням графічної інформації на екран запускається процедура прогону алгоритму на існуючих наборах вхідних даних зі збором необхідної статистичної інформації | Високий |
| Адміністратор | Редагування інформації бази даних | Адміністратор має безпосередній доступ для редагування інформації бази даних системи | Високий |
| Генерація наборів вхідних даних | Адміністратор запускає процедуру генерації набору вхідних даних за введеними параметрами | Середній |
| Накопичення оптимальних розв’язків | Адміністратор може накопичувати оптимальні розв’язки для наборів вхідних даних | Високий |

## **Огляд наявних аналогів**

Під час розробки будь-якого програмного забезпечення, одним з етапів його проектування є пошук вже існуючих рішень. Для оцінки доцільності створення нового програмного забезпечення необхідно виявити всі переваги та недоліки існуючих систем.

Програмний продукт, що розроблюється, базується на ряді наукових робіт в області важкорозв’язуваних задач побудови розкладів. Фактично, був створений новий метод розв’язання такого роду задач – ПДС-алгоритми, аналогів якому не існує в світі.

Хоча дана предметна область є вузькою, швидкий розвиток теорії розкладів створює потребу у такого роду системі. І хоча на сьогоднішній день не існує системи, яка б повністю задовольнила всі висунуті вимоги, дана система буде включати та поєднувати функції, що доступні в окремих системах, таких як системи контролю версій.

Наприклад, новий програмний продукт буде включати базові функції системи контролю версій. Ілюстрацію використання однієї з таких систем – github, наведено на наступному рисунку:

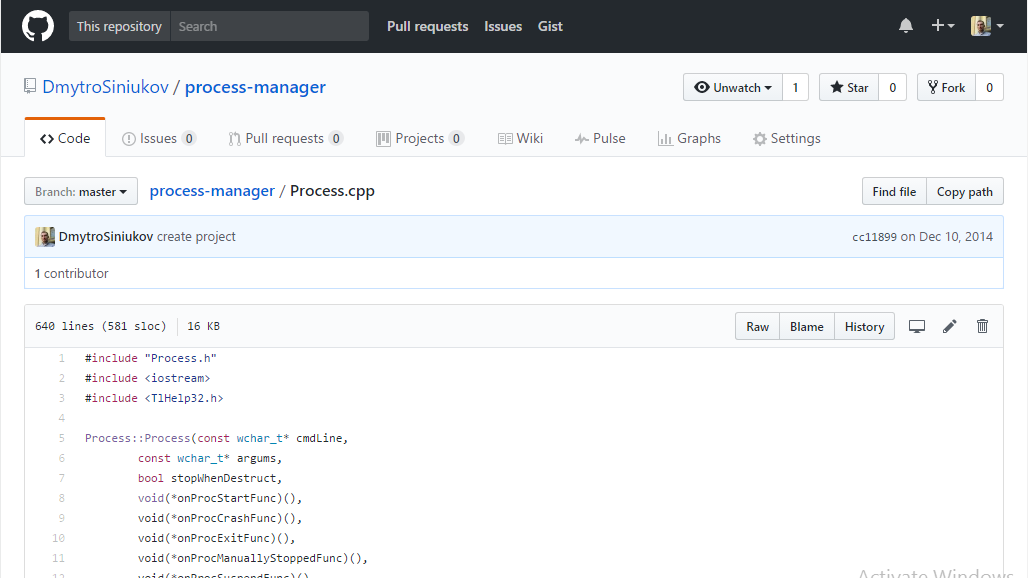


Рисунок 1.1 – Форма редагування вихідного коду у системі github

Крім того, система також включає функції онлайн інтерпретаторів коду, що дозволяють ввести і запустити користувацьку програму в режимі реального часу, не перезапускаючи весь проект.

Але, дана система має і унікальні функції, що потребуються для розробки алгоритмів саме ПДС-класу. Наприклад, аналіз ефективності поліноміальної та декомпозиційної складових ПДС-алгоритму.

Беручи до уваги складність побудови алгоритмів вирішення задач складання розкладів а також те, що сьогодні є високий попит в ефективному вирішенні таких задач (наприклад для планування робіт на виробництві), є потреба в інформаційній системі, яка б допомагала науковцям у спільній   
розробці ПДС-алгоритмів.

Зважаючи на те, що новий програмний продукт буде задовольняти попит, а також, що на сьогодні не існує системи, яка б одночасно включала всі необхідні функції, розробка даної системи є актуальною. Система буде ефективно поєднувати в собі комплекс функцій, що доступні у розрізнених системах, а також задовольняти вимоги, притаманні саме для розробки ПДС-алгоритмів.

## **Постановка задачі**

### **Призначення розробки**

Система підтримки розробки та аналізу ПДС-алгоритмів побудови розкладів робіт на паралельних верстатах однакової продуктивності призначена для інформаційно-аналітичного забезпечення процесів розробки ПДС-алгоритмів складання розкладів.

### **Цілі та задачі розробки**

Основними цілями розробки системи підтримки розробки та аналізу ПДС-алгоритмів побудови розкладів робіт на паралельних верстатах однакової продуктивності є:

* створення комфортних умов для ефективної командної розробки ПДС-алгоритмів;
* підвищення ефективності процесів розробки, тестування та аналізу ПДС-алгоритмів;
* підвищення швидкості розробки, тестування та аналізу ПДС-алгоритмів.

Для досягнення поставлених цілей мають бути вирішені такі задачі:

* забезпечення інтерпретації та виконання вихідного коду алгоритмів в режимі реального часу;
* представлення статистичної інформації аналізу алгоритмів у наглядному (графічному) вигляді;
* ефективне розділення прав користувачів різних груп: неавтентифіковані користувачі, розробники алгоритмів, адміністратори;
* реалізація точного алгоритму знаходження оптимальних розв’язків;
* ведення інформації про ПДС-алгоритми задач складання розкладів, включаючи вихідний код алгоритмів;
* ведення наборів вхідних даних для тестування та аналізу;
* накопичення оптимальних розв’язків;
* автоматичне тестування та аналіз алгоритмів;
* побудова розкладу обраним алгоритмом для довільних введених вхідних даних.

## **Висновок до розділу**

В даному розділі було описано предметне середовище нової системи та було дано визначення основних понять, що будуть використовуватись у наступних розділах.

Також, було розглянуто процес діяльності, включаючи типові дії, що виконує головний користувач – розробник ПДС-алгоритмів, під час розробки, тестування та аналізу нового алгоритму. Крім того, в даному розділі було наведено функціональну структуру системи з її акторами і основними функціями, що їм доступні.

В даному розділі було описано існуючі системи-аналоги. Було наведено деякі типи систем, що мають спільні функції.

У заключному підрозділі було описано постановку задачі. Було наведено призначення розробки, цілі та задачі розробки.

# Інформаційне забезпечення

## **Вхідні дані**

### **Аналітики**

Аналітики вносяться у систему адміністратором вручну через відповідну екранну HTML-форму . Дані передаються на сервер застосунку в тілі HTTP-запиту.

Таблиця 2.1 – Опис параметрів аналітики

|  |  |  |
| --- | --- | --- |
| **Назва параметра** | **Опис** | **Тип даних** |
| Назва | Назва аналітики | Рядок |
| Крок | Числовий інтервал на осі X графіку аналітики | Дійсне число |
| Опис | Детальний опис аналітики | Текст |
| Код функції | Вихідний код залежності значення ординати від вхідних даних і поточного алгоритму у заданому форматі | Текст |
| Мітка осі X | Надпис на осі абсцис графіку | Рядок |
| Мітка осі Y | Надпис на осі ординат графіку | Рядок |
| Максимальне значення | Найбільше значення по осі ординат | Дійсне число |

### **Набори вхідних даних аналітик**

Набори вхідних даних аналітик вносяться у систему адміністратором у вигляді текстового файлу наступного формату:

,;…;,

,;…;,

…

,;…;,,

де – кількість верстатів рядку вхідних даних ; , – відповідно довжина і директивний термін завдання рядку вхідних даних .

Приклад:

3 2,3;2,4;3,5;1,6;2,8

3 1,3;1,4;3,6;2,7;1,9;6,15

5 1,3;1,4;2,5;1,6;2,8.

### **Алгоритми**

Алгоритми вносяться у систему користувачем (розробником алгоритмів) вручну через відповідну екранну HTML-форму. Дані передаються на сервер застосунку в тілі HTTP-запиту.

Таблиця 2.2 – Опис параметрів алгоритму

|  |  |  |
| --- | --- | --- |
| **Назва параметру** | **Опис** | **Тип даних** |
| Назва | Назва алгоритму | Рядок |
| Вихідний код | Вихідний код процедури формування розкладу | Текст |
| Опис | Детальний опис алгоритму у вільному форматі | Текст |
| Опублікований | Ознака публікації алгоритму | Булевий |

### **Вхідні дані для побудови розкладу**

Розробник алгоритмів має можливість випробувати доступний йому алгоритм (його власний або опублікований алгоритм іншого користувача), побудувавши розклад для довільних введених ним вхідних даних. Вхідні дані передаються у систему у вигляді двох файлів: файлу з описом верстатів і файлу з описом завдань.

Формат вхідного файлу верстатів:

*m*

*[назва верстату 1]*

…

*[назва верстату m],*

де – кількість верстатів; – локальні ідентифікатори верстатів у межах даного розкладу. Назви та ідентифікатори верстатів використовуються для більш дружнього для користувача представлення розкладу на діаграмі Ґанта.

Приклад:

3

1 “Machine 1”

2 “Machine 2”

3 “Machine 3”.

Формат вхідного файлу завдань:

*n*

*[назва завдання 1]*

…

*[назва завдання n] ,*

де – кількість завдань; – локальні ідентифікатори завдань у межах даного розкладу; – тривалості завдань; – директивні терміни завдань. Назви та ідентифікатори завдань використовуються для більш дружнього для користувача представлення розкладу на діаграмі Ґанта.

Приклад:

5

1 “Task 1” 2 7

2 “Task 2” 7 19

3 “Task 3” 6 13

4 “Task 4” 8 14

5 “Task 5” 2 8.

## **Вихідні дані**

Основними вихідними даними є розклади робіт, побудовані користувацькими алгоритмами, а також точним системним алгоритмом. Також, система виводить графіки аналізу ефективності алгоритмів і діаграми Ґанта розкладів робіт, що були побудовані обраним алгоритмом на довільних вхідних даних. Приклади таких графіків наведені на рисунках 2.1, 2.2:

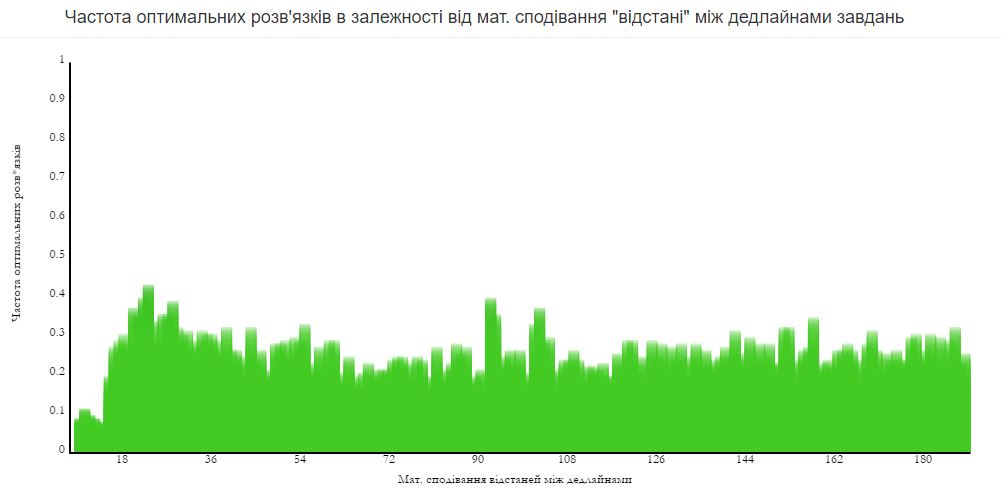


Рисунок 2.1 – Графік однієї з аналітик алгоритму

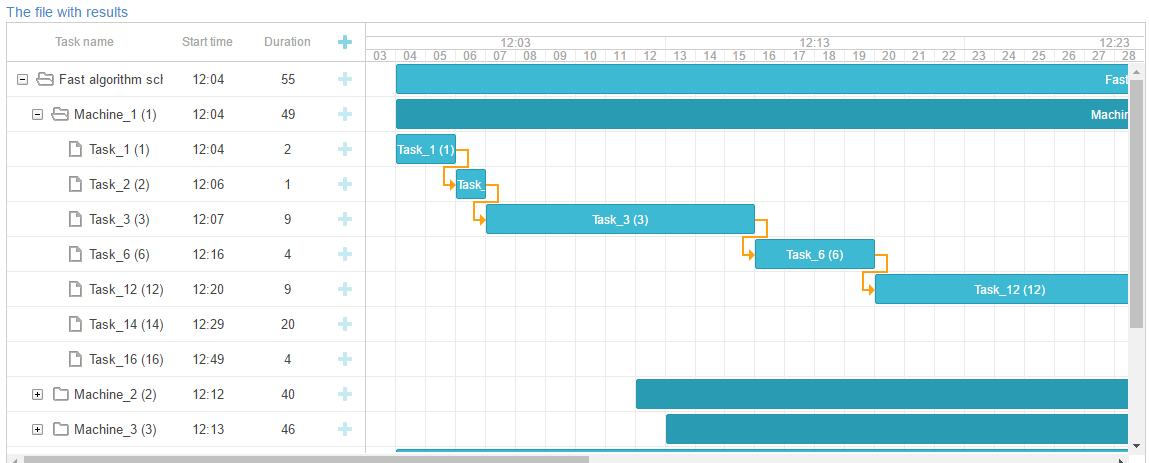


Рисунок 2.2 – Діаграма Ґанта побудованого розкладу робіт

## **Опис структури бази даних**

Фізичну діаграму структури основної бази даних системи наведено у частині графічного матеріалу дипломного проекту. Далі наведено опис основних сутностей і їх атрибутів.

Таблиця 2.3 – Опис сутностей бази даних

|  |  |
| --- | --- |
| **Назва сутності** | **Опис** |
| Користувач (User) | Представляє користувача системи |
| Роль (Role) | Описує роль користувача системи (розробник алгоритмів, адміністратор) |
| Роль користувача (UserRole) | Таблиця-зв’язка для зв’язку «багато до багатьох» між користувачами і ролями |
| Алгоритм (Algorithm) | Містить інформацію про ПДС-алгоритм, включаючи його вихідний код |
| Аналітика (Analytic) | Сутність, що представляє дані для аналізу довільного алгоритму за однією ознакою |
| Вхідні дані (Input) | Представляє вхідні дані аналітики для одного запуска алгоритму |
| Аналітика алгоритму (AnalyticAlgorithm) | Сутність, що представляє кешовані дані аналізу конкретного алгоритму |

Таблиця 2.4 – Опис атрибутів сутності «Користувач»

|  |  |
| --- | --- |
| **Назва атрибута** | **Опис** |
| Ім’я користувача (Login) | Псевдоним користувача у системі |
| Повне ім’я (FullName) | Повне ім’я (наприклад прізвище, ім’я та по-батькові) |
| Електронна пошта (Email) | Адреса електронної пошти користувача |
| Дата народження (BirthDate) | Дата народження користувача |

Продовження таблиці 2.4

|  |  |
| --- | --- |
| **Назва атрибута** | **Опис** |
| Хеш-код пароля (PasswordHash) | Захешоване значення пароля користувача |
| Відбиток безпеки (SecurityStamp) | Використовується для підтримки безпеки інформації в системі |
| Кількість невдалих спроб входу в систему (AccessFailedCount) | Кількість невдалих спроб входу в систему (для блокування облікового запису при загрозі взламу) |

Таблиця 2.5 – Опис атрибутів сутності «Алгоритм»

|  |  |
| --- | --- |
| **Назва атрибута** | **Опис** |
| Назва (Name) | Назва алгоритму |
| Вихідний код (Code) | Вихідний код процедури формування розкладу |
| Опис (Description) | Опис алгоритму у вільному форматі |
| Дата створення (DateAdd) | Дата створення алгоритму у системі |
| Опублікований (Published) | Ознака публікації алгоритму |

Таблиця 2.6 – Опис атрибутів сутності «Аналітика»

|  |  |
| --- | --- |
| **Назва атрибута** | **Опис** |
| Назва (Name) | Назва аналітики |
| Дата створення (DateAdd) | Дата створення аналітики в системі |
| Крок (Step) | Числовий інтервал на осі X графіку аналітики |
| Опис (Description) | Опис аналітики у вільному форматі |
| Код функції (FunctionCode) | Вихідний код залежності значення ординати від вхідних даних і поточного алгоритму |
| Мітка осі X (XLabel) | Надпис на осі абсцис графіку |
| Мітка осі Y (YLabel) | Надпис на осі ординат графіку |

Продовження таблиці 2.6

|  |  |
| --- | --- |
| **Назва атрибута** | **Опис** |
| Максимальне значення (MaxValue) | Найбільше значення по осі ординат |

Таблиця 2.7 – Опис зв’язків між сутностями

|  |  |
| --- | --- |
| **Ідентифікатор зв’язку** | **Опис** |
| FK\_UserRole\_Role, FK\_UserRole\_User | Зв’язок «багато до багатьох» між користувачами і ролями (користувач може мати декілька ролей) |
| FK\_Algorithm\_User | Користувач, що створив алгоритм |
| FK\_Analytic\_User | Користувач, що створив аналітику |
| FK\_Input\_Analytic | Аналітика, до якої відноситься рядок вхідних даних |
| FK\_AnalyticAlgorithm\_Analytic | Зв’язок між аналітикою та її кешованими даними аналізу алгоритмів |
| FK\_AnalyticAlgorithm\_Algorithm | Зв’язок між алгоритмом та його кешованими даними аналізу |

## **Висновок до розділу**

Розділ інформаційного забезпечення присвячений опису системи з точки зору даних, які у ній зберігаються та оброблюються. В даному розділі було описано вхідні дані і документи з інформацією, що надходять у систему. Крім того, було описано вихідні дані, які система видає користувачу для аналізу, включаючи графічну інформацію. В заключному підрозділі було докладно описано структуру база даних, її сутності, зв’язки і атрибути.

Таким чином, в даному розділі було проведено аналіз системи в розрізі інформаційного забезпечення.

# Математичне забезпечення

## **Змістовна постановка задачі**

Існує набір незалежних паралельних приладів з рівними продуктивностями, що працюють без переривань та виконують незалежні завдання з довільними фіксованими директивними термінами та довільними фіксованими тривалостями виконання. Завдання мають бути виконані до директивних термінів. Моменти запуску приладів довільні.

Необхідно знайти допустимий (за директивними термінами) розклад, у якому кожний прилад має момент запуску якомога пізніший, з урахуванням векторного (лексикографічного) порядку.

## **Математична постановка задачі**

Нехай маємо *m* незалежних паралельних верстатів однакової продуктивності, які працюють без переривань і виконують *n* робіт ( – тривалість виконання -ї роботи, ). Роботи повинні бути виконані до директивних термінів . Моменти запуску верстатів довільні. Необхідно побудувати допустимий розклад, що максимізує наступний критерій:

|  |  |  |
| --- | --- | --- |
|  |  | (3.1) |

де – момент запуску верстата , – номер верстату, у якого момент запуску в оптимальному розкладі найбільш ранній (він є найбільш пізнім для всіх допустимих розкладів); , – номер верстату, у якого момент запуску наступний за величиною після верстатів , (він є найбільш пізнім для всіх допустимих розкладів з фіксованим ).

## **Обґрунтування методу розв’язання**

Математична модель задачі про оптимізацію розкладу робіт з довільними директивними термінами на паралельних приладах рівних продуктивностей за лексикографічним критерієм належить до задач багатокритеріального програмування.

Було виділено два основних методи розв’язання даної задачі: ПДС-алгоритм і алгоритм, що реалізує метод гілок та меж для даної задачі складання розкладу.

ПДС-алгоритм, що складається з двох частин: поліноміальної точної складової та евристичної наближеної складової, є ефективним, як за точністю розв'язку, так і за швидкістю. Головною ціллю є саме дослідження його ефективності та можливості вдосконалення. Основна ідея ПДС-алгоритму полягає у наступному: спочатку алгоритм намагається побудувати оптимальний розклад, перевіряючи деякі ознаки оптимальності, якщо такий розклад побудувати не вдалося (ознаки не спрацювали, або допустимий розклад при виконанні даних ознак не було знайдено), то розв’язок знаходиться евристичними алгоритмами, обирається найкращий з отриманих результатів. Проте цей метод не дає точного результату у ста відсотках випадків, тому для отримання оптимального розкладу було використано метод гілок та меж.

Метод гілок та меж є широко відомим методом дискретної оптимізації. Він зазвичай використовується для розв’язання NP-повних задач або для яких точний метод розв’язання не є відомим. Основна ідея полягає у розумному переборі множини розв’язків з відкиданням завідомо неоптимальних розв’язків. Основні операції: розбиття головної множини розв’язків на підмножини, знаходження оцінки, збереження рекорду, тест. В загальному випадку асимптотична складність алгоритму є експоненціальною (в найгіршому випадку алгоритм проходить всі можливі варіанти розкладів) [6].

## **Опис методів розв’язання**

### **Опис ПДС-алгоритму розв’язання задачі**

Нехай – деяка константа, яка буде обмежувати глибину перебору для збереження поліноміальної залежності часу роботи алгоритму від числа .

Відсортуємо завдання за спаданням чисел .

Робота алгоритму складається з набору однотипних ітерацій, на кожній з яких призначається чергове завдання на один з вже виділених верстатів, якщо це можливо так, щоб не змінився час запуску даного об’єкта. Інакше, перевіряються умови оптимальності початкового розкладу в разі виділення вільного верстата.

Ітерації продовжуються до тих пір, доки не виконається одна з умов:

а) на кожен із верстатів призначене хоча б одне завдання;

б) всі завдання призначені на виконання;

в) поточне завдання не виходить призначити ні на один з виділених верстатів так, щоб початковий розклад залишався допустимим, при цьому жодна з умов виділення нового верстата не виконується.

На першій ітерації призначається завдання з індексом на верстат з індексом із початком виконання в момент часу .

Нехай після ітерації :

– множина індексів усіх завдань, призначених на виконання, – множина індексів виділених верстатів, – множина індексів завдань, призначених на виконання на верстат з індексом , тоді = , , .

Після кожної ітерації виконується:

|  |  |  |
| --- | --- | --- |
|  |  | (3.1) |

Позначимо множину індексів виділених верстатів, що є допустимими для поточного завдання на ітерації , . Множина на поточній ітерації визначається наступним чином: для кожного об'єкта з індексом з множини намагаємося вставити поточне завдання на своє місце за зростанням директивних термінів. При цьому завдання верстата, що перевіряється, які розташовані пізніше цього місця, посуваються блище до своїх директивних термінів на величину .

Нехай , де – множина індексів завдань, що були посунуті, верстата, який на даний момент перевіряється, – момент закінчення виконання завдання з індексом .

Тоді:

|  |  |  |
| --- | --- | --- |
|  |  | (3.2) |

Де , , якщо (або на ітерації спрацювала умова поновлення однозначності), інакше .

Нехай – множина «неоднозначних» завдань на ітерації :

|  |  |  |
| --- | --- | --- |
|  |  | (3.3) |

На ітерації *i ≠ 1* виконуємо наступну процедуру:

1. Перевіряємо умови *а, б* закінчення роботи алгоритму. Якщо одна з них спрацювала – початковий розклад було побудовано, алгоритм завершує свою роботу. В іншому випадку – переходимо до пункту 2.

2. Якщо , призначаємо поточне завдання на один з верстатів (будь-який), якому відповідає мінімум: , на позицію за зростанням директивних термінів, переходимо до пункту 3. В іншому випадку –до пункту 4.

3. Якщо , строго перебираємо всі варіанти призначень з обмеженням глибини перебору. Якщо процедура показала, що існує тільки один варіант призначень (поточний) – умова відновлення однозначності виконується для поточної ітерації, очищуємо множини . Переходимо до наступної ітерації.

4. Якщо , виділяємо черговий об'єкт, призначаємо на нього поточне завдання з початком виконання в момент , переходимо до наступної ітерації. В іншому випадку переходимо до пункту 5.

5. Перевіряємо умову виділення вільного об'єкта. Якщо умова виконується, виділяємо черговий об'єкт, призначаємо на нього завдання *i* з початком виконання в момент часу , видаляємо *i* з множини , переходимо до наступної ітерації. Інакше переходимо до пункту 6.

6. Якщо – початковий розклад для поточного набору вхідних даних побудувати не вдалося, алгоритм завершує свою роботу. В іншому випадку виконуємо процедуру строгого перебору варіантів призначень з обмеженням глибини занурення, переходимо до наступної ітерації.

### **Експериментальне дослідження обчислювальної складності алгоритмів**

На рисунках 3.1, 3.2 наведено графіки залежності часу виконання алгоритмів від розміру вхідних даних.

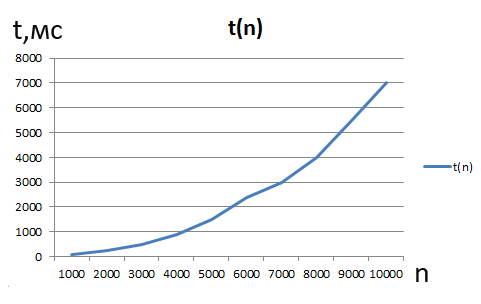


Рисунок 3.1 – Графік залежності часу виконання ПДС-алгоритму від кількості завдань

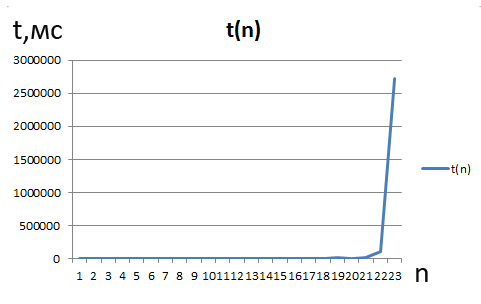


Рисунок 3.2 – Графік залежності часу виконання процедури методу гілок та меж від кількості завдань

## **Висновок до розділу**

Даний розділ був присвячений математичній складовій проекту. Було описано два основних методи розв’язання даної задачі складання розкладів –   
  
ПДС-алгоритм і модифікований алгоритм методу гілок та меж. Було проаналізовано обчислювальну складність методів розв’язання задачі в залежності від розміру вхідних даних.

# Програмне та технічне забезпечення

## **Засоби розробки**

Основними засобами розробки системи є платформа .NET і мова програмування C#. За середовище розробки було взято Visual Studio в поєднанні з системою контролю версій git і інструментами веб-розробки від компанії Майкрософт (Microsoft).

C# – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Синтаксис C# близький до С++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато що від своїх попередників — мов С++, Delphi, Модула і Smalltalk — С#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад множинне спадкування класів (на відміну від C++) [7].

Головними перевагами вказаних засобів розробки для даної системи є високий рівень абстракції і багатство математичних функцій у стандартних бібліотеках. Це допомагає в розробці систем математичної спрямованості, якою є дана.

Веб-інтерфейс системи реалізований з використанням технології Майкрософт ASP.NET MVC.

Для зручної і ефективної роботи з даними, в даній системі використовується технологія об’єктно-реляційного відображення, яку реалізує Entity Framework.

## **Вимоги до технічного забезпечення**

### **Загальні вимоги**

Клієнтський застосунок даної системи створений у вигляді веб-сайту. Для користування системою користувач повинен мати стабільне інтернет-з’єднання, а також основні засоби доступу до мережі Інтернет.

## **Архітектура програмного забезпечення**

Система є веб-застосунком з архітектурою Model-View-Controller (модель-вид-контролер).

Модель-вид-контролер – архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення. Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача. Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності [8].

### **Діаграма класів**

В даному пункті описано найбільш значущі класи системи, повну діаграму класів наведено у частині графічного матеріалу.

Для реалізації архітектури модель-вид-контролер були створені класи двох типів: класи-моделі – класи, що представляють форми даних, класи-контролери – класи, що приймають інформацію у вигляді моделей з представлень, оброблюють її і повертають результат користувачу у вигляді представлення.

Наступна таблиця описує класи-моделі.

Таблиця 4.1 – Опис класів-моделей

|  |  |
| --- | --- |
| **Назва класу** | **Опис** |
| Користувач (ApplicationUser) | Об’єкти класу представляють інформацію про поточного користувача |
| Алгоритм (Algorithm) | Інформація про алгоритм з його вихідним кодом |
| Вхідні дані (Input) | Представляє один рядок вхідних даних |
| Аналітика (Analytic) | Інформація для аналізу довільного алгоритму |
| Аналіз алгоритму (AlgorithmAnalysisModel) | Повна інформація для аналізу алгоритму |

Таблиця 4.2 – Опис класів-контролерів

|  |  |
| --- | --- |
| **Назва класу** | **Опис** |
| Домашній (HomeController) | Оброблює інформацію, що доступна всім користувачам |
| Управлінський (ManageController) | Оброблює інформацію, що доступна лише авторизованим користувачам |

Таблиця 4.3 – Опис основних службових класів

|  |  |
| --- | --- |
| **Назва класу** | **Опис** |
| Репозиторій (Repository) | Клас, що надає інтерфейс програмного доступу до бази даних |
| Службові методи (ServiceMethods) | Набір службових методів для обробки даних |

Продовження таблиці 4.3

|  |  |
| --- | --- |
| **Назва класу** | **Опис** |
| Розклад (Schedule) | Клас, що представляє розклад, а також програмний інтерфейс для роботи з розкладами |

### **Діаграма послідовності**

У графічному матеріалі наведено діаграму послідовності для варіанту використання «аналіз алгоритму».

Розробник алгоритмів надсилає запит до контролера «ManageController» на отримання інформації аналізу алгоритму, передаючи ідентифікатор алгоритму в параметрах посилання. Контролер оброблює запит і, в свою чергу, викликає метод репозиторія на отримання необхідних даних. В тому випадку, якщо аналіз даного алгоритму ще не відбувався або алгоритм був змінений, відбувається запит на отримання додаткових даних для побудови аналітик алгоритму. Робиться виклик методу побудови графіків, графічна інформація повертається користувачу як результат запиту.

### **Діаграма компонентів**

Зважаючи на те, що дана система є веб-застосунком архітектури модель-вид-контролер, система включає основні компоненти даного типу архітектури: контролери, моделі і представлення.

Окрім «обов’язкових» компонентів, наведено і компоненти, характерні для даної конкретної системи: інфраструктура – програмний інтерфейс зі службовими методами для обчислень і роботи з базою даних, логіка побудови розкладів – користувацька бібліотека з алгоритмами побудови розкладів, Entity Framework – стандартна бібліотека, що реалізує технологію об’єктно-реляційного відображення.

Діаграму компонентів наведено у розділі графічного матеріалу.

### **Специфікація функцій**

В даному пункті описані ключові методи класів системи.

Таблиця 4.4 *–* Специфікація функцій

| **Клас** | **Метод** | **Параметри** | **Опис** |
| --- | --- | --- | --- |
| Repository | GetAvailableAlgorithms | string userId | Отримати алгоритми, доступні користувачу |
| GetAlgorithm | int id | Отримати алгоритм за ідентифікатором |
| UpdateAlgorithm | Algorithm algorithm | Оновити алгоритм в базі даних |
| InsertInput | Input input | Зберегти вхідну інформацію в БД |
| GetAnalytics | void | Отримати аналітики |
| GetInputs | void | Отримати вхідні дані, «розібрані» по аналітикам |
| Schedule | BuildOptimalSchedule | IEnumerable<Task> tasks, IEnumerable<Machine> machines | Побудувати оптимальний розклад |
| BuildSchedule | IEnumerable<Task> tasks, IEnumerable<Machine> machines | Побудувати розклад за основним ПДС-алгоритмом |

Продовження таблиці 4.4

|  |  |  |  |
| --- | --- | --- | --- |
| ManageController | Algorithms | void | Отримати відображення доступних користувачу алгоритмів |
| CreateAlgorithm | void | Отримати відображення форми додавання алгоритму |
| EditAlgorithm | int id | Отримати відображення форми редагування алгоритму |
| EditAlgorithm | Algorithm algorithm | Обробити дані алгоритму, внесені користувачем |
| DeleteAlgorithm | int id | Видалити алгоритм, отримати відображення попередньої сторінки |
| AlgorithmAnalysis | int id | Отримати відображення сторінки аналізу алгоритму |

## **Висновок до розділу**

Даний розділ дав докладну характеристику програмного та технічного забезпечення системи, описавши засоби розробки, вимоги до технічного забезпечення та архітектуру програмного забезпечення.

# Технологічний розділ

В даному розділі буде наведено керівництво користувача, описано використання основних функцій системи розробником алгоритмів – створення облікового запису, авторизація, створення та редагування алгоритмів, тестування та аналіз алгоритмів. Випробування системи будуть описані в додатках до дипломного проекту.

## **Керівництво користувача**

1. Початок роботи із системою. Головна сторінка веб-застосунку має наступний вигляд:

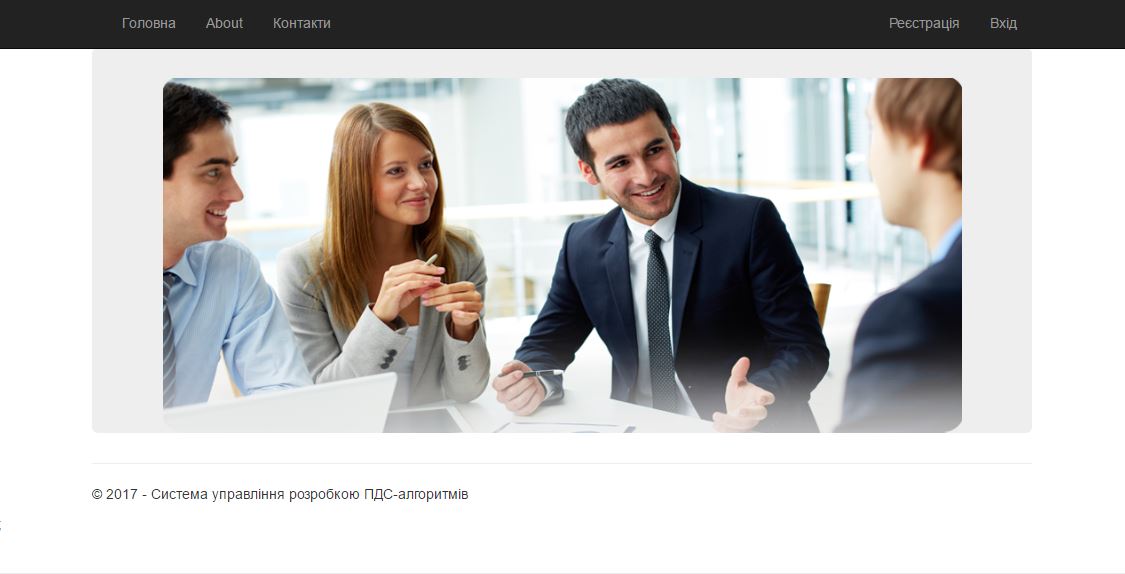


Рисунок 5.1 – Головна сторінка системи

1. Для роботи в системі новий користувач має створити обліковий запис. За замовчуванням система надає новому користувачу роль розробника алгоритмів. Сторінка із формою створення облікового запису зображена на рисунку 5.2:

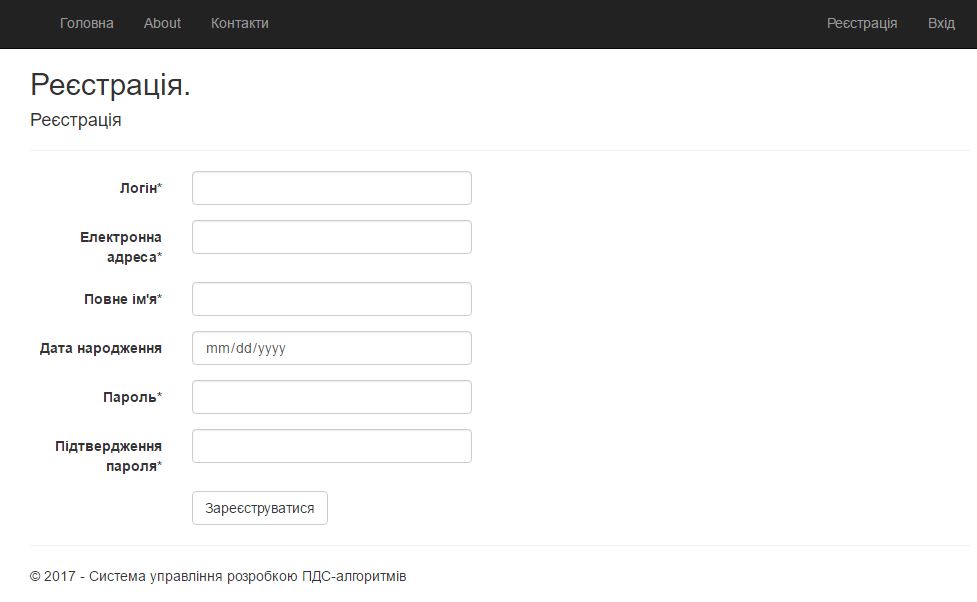


Рисунок 5.2 – Форма створення облікового запису

Заповнюємо форму даними користувача:

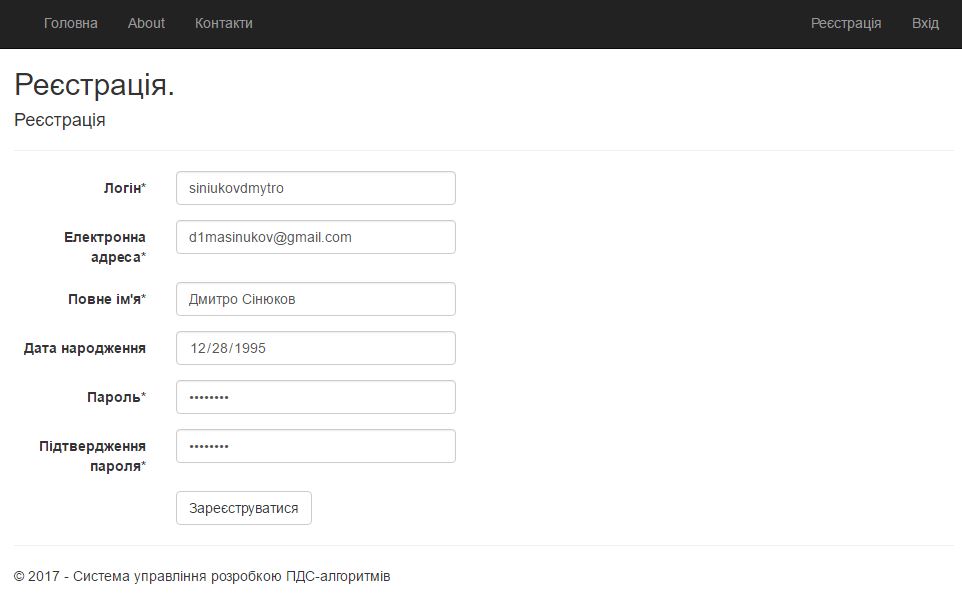


Рисунок 5.3 – Заповнена форма реєстрації

1. Після успішного створення облікового запису необхідно виконати вхід в систему:

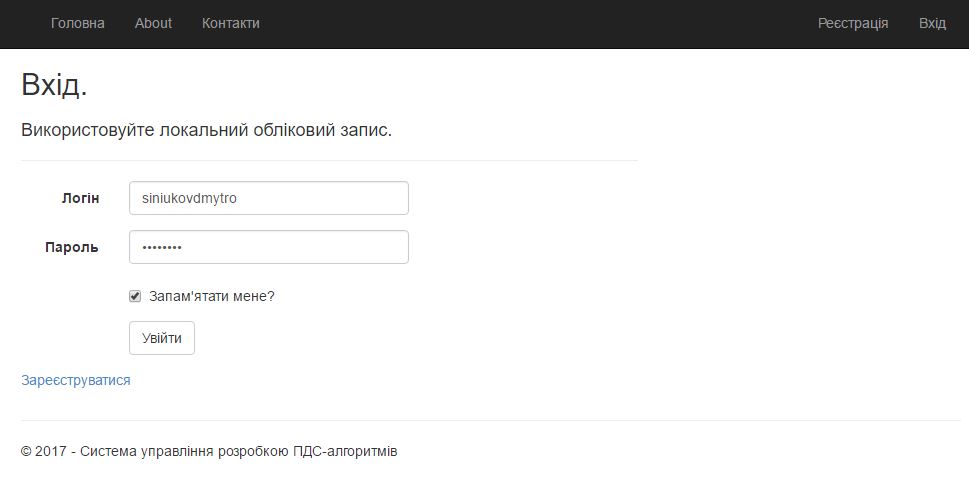


Рисунок 5.4 – Вхід в систему

1. Після входу в систему на головній панелі з’являються кнопка із надписом привітання і кнопка виходу з облікового запису. Натиснувши на кнопку з привітанням, система перенаправляє користувача на сторінку особистого кабінету:

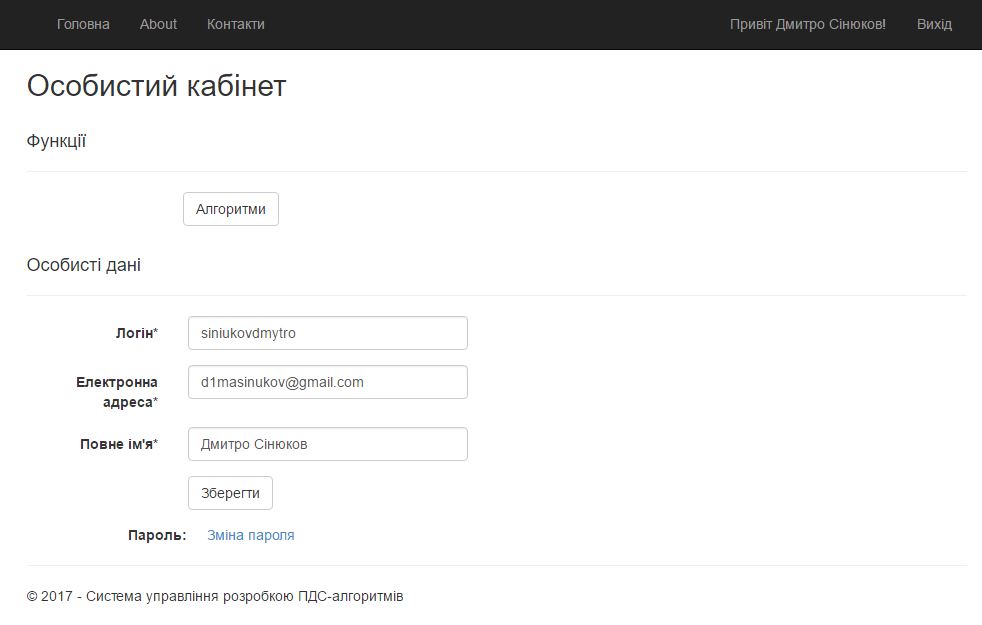


Рисунок 5.5 – Особистий кабінет розробника алгоритмів

1. Робота користувача в ролі розробника починається після натискання кнопки «Алгоритми». Користувач бачить доступні йому алгоритми:

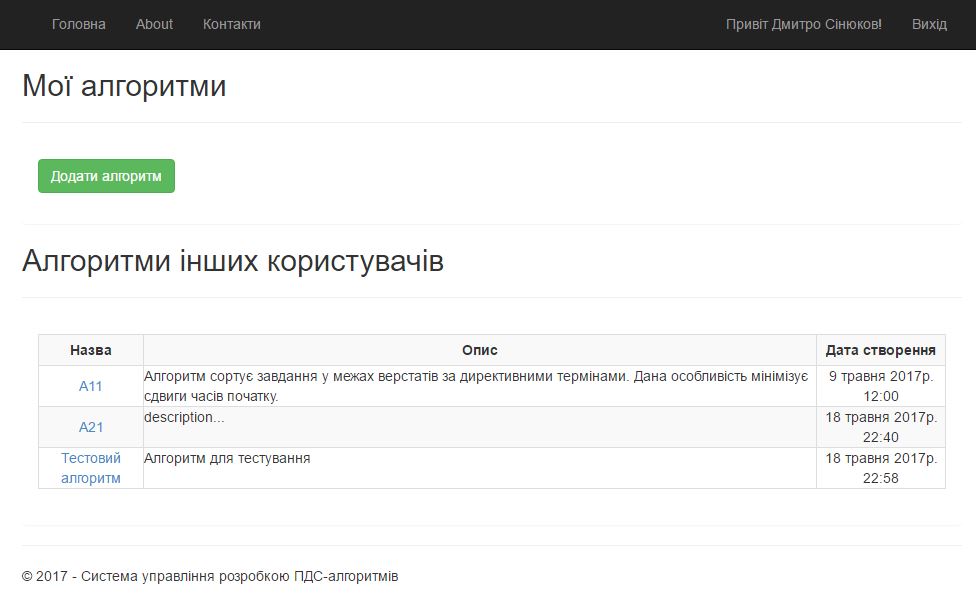


Рисунок 5.6 – Сторінка алгоритмів

1. На даний момент користувач не має його власних алгоритмів. Створимо новий алгоритм, натиснувши кнопку «Додати алгоритм» і заповнивши поля форми коректними даними:

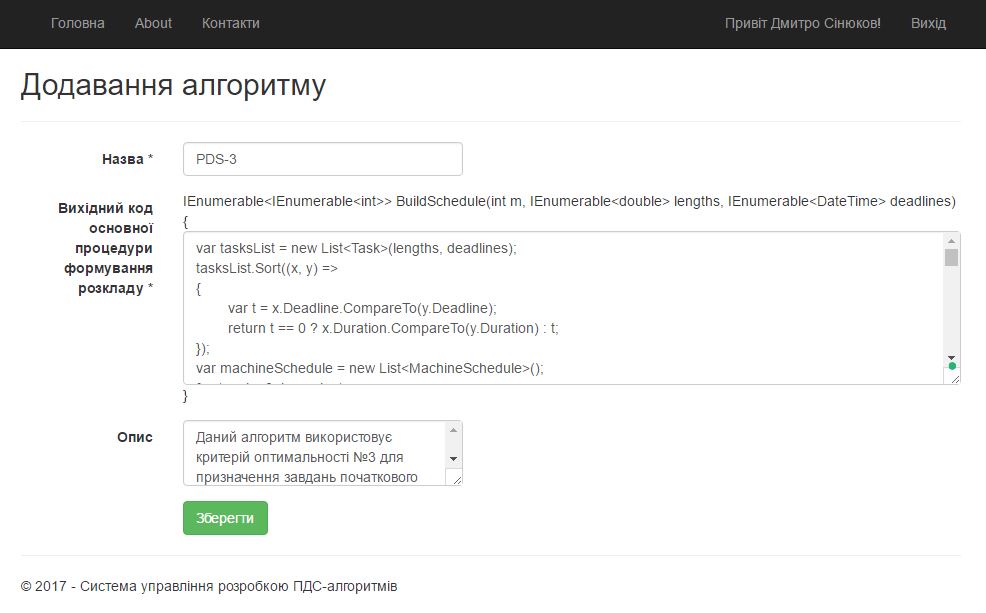


Рисунок 5.7 – Заповнена форма додавання алгоритму

Як виявилося, вихідний код алгоритму має помилки компіляції, система інформує про це користувача і відхиляє збереження алгоритму:

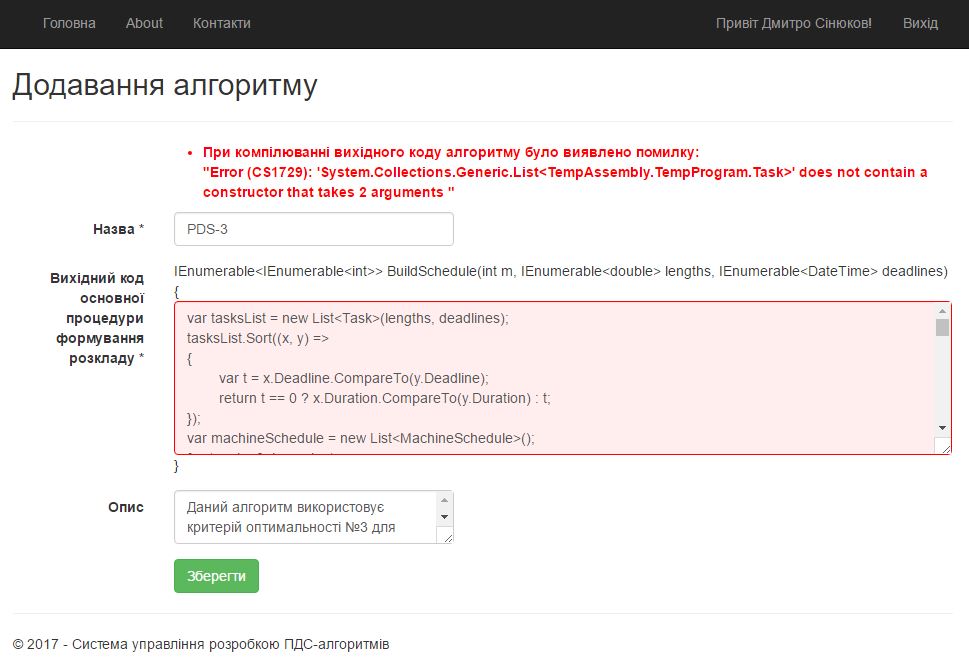


Рисунок 5.8 – Повідомлення про помилку компіляції

Після виправлення помилки компіляції, система виявила помилку при тестуванні алгоритму на тестовому наборі даних:

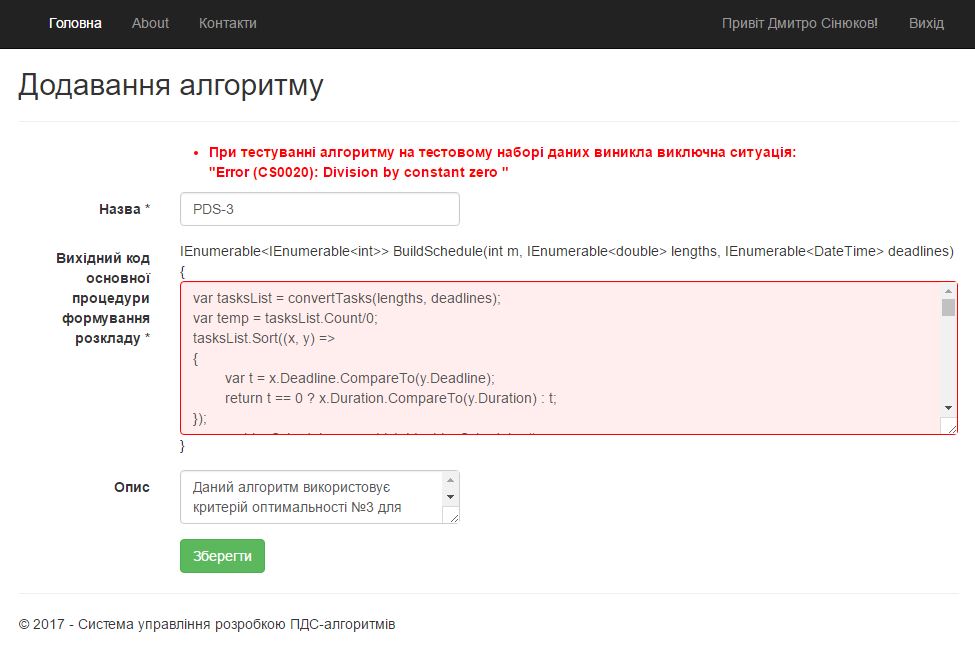


Рисунок 5.9 – Повідомлення про помилку при тестуванні алгоритму

Користувач виправляє помилки вихідного коду алгоритму. Після успішного збереження алгоритму, останній стає доступним серед основного списку алгоритмів:

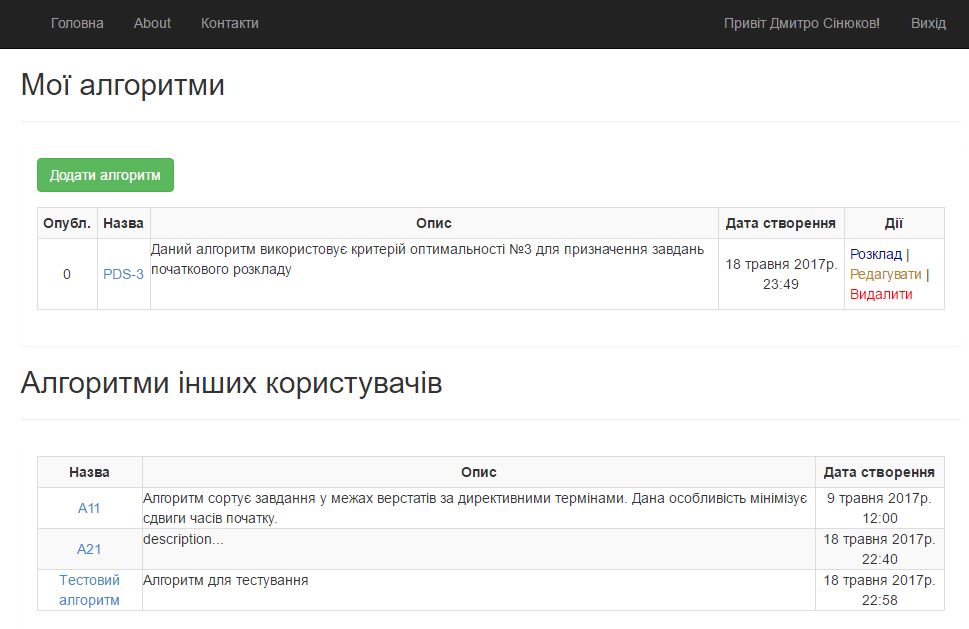


Рисунок 5.10 – Сторінка алгоритмів після додавання нового алгоритму

1. Для збору статистичної інформації про алгоритм і аналізу ефективності алгоритму за доступними аналітиками необхідно перейти на сторінку аналізу алгоритму, натиснувши посилання в таблиці з назвою алгоритму. Сторінка аналізу алгоритму:

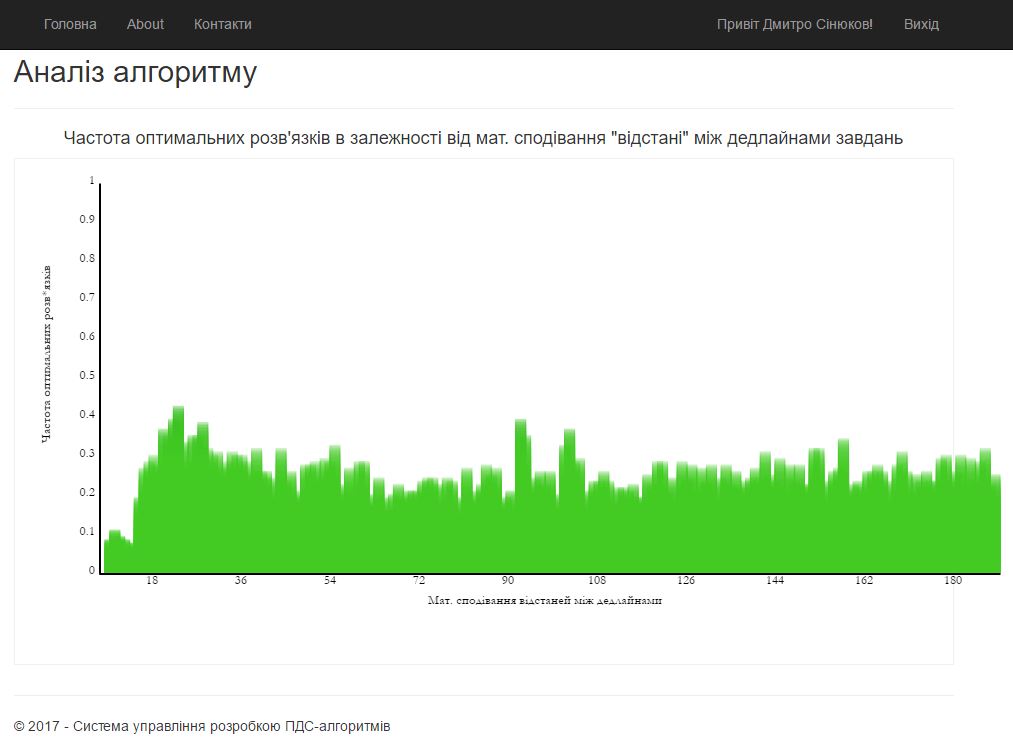


Рисунок 5.11 – Сторінка аналізу алгоритму

Сторінка аналізу є динамічною, тому що залежить від інформації, що зберігається в базі даних системи, а саме: даних таблиць аналітик і вхідних даних. Доступ до зміни цих даних мають користувачі ролі «Адміністратор».

## **Випробування програмного продукту**

За проектним планом у системі наявні три актори: неавтентифікований користувач, розробник алгоритмів і адміністратор.

Згідно з проектом можна виділити 4 основних сценарії. До них входять наступні варіанти використання:

* авторизація;
* додавання алгоритму;
* аналіз алгоритму;
* побудова розкладу.

### **Мета випробувань**

Метою випробувань є перевірка працездатності функцій системи, її відповідність винесеним вимогам, що описані в технічному завданні до розробки системи.

### **Загальні положення**

Випробування проводяться на основі наступних документів:

* ГОСТ 34.603−92. Інформаційна технологія. Види випробувань автоматизованих систем;
* ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

### **Результати випробувань**

У процесі тестування була перевірена основна функціональність WEB-застосування. У наступних таблицях наведений перелік випробувань основних функціональних можливостей (таблиці 5.1 – 5.4).

Таблиця 5.1 – Авторизація

| **Мета тесту:** | **Перевірка функції «Вхід в систему»** |
| --- | --- |
| Початковий стан ПЗ | Відкрито сторінку авторизації |
| Вхідні дані: | Логін і пароль користувача |
| Схема проведення тесту: | Заповнити поля форми авторизації, натиснути кнопку «Вхід» |

Продовження таблиці 5.1

|  |  |
| --- | --- |
| **Мета тесту:** | **Перевірка функції «Вхід в систему»** |
| Очікуваний результат: | Відкрито головну сторінку системи, на головній панелі з’явилася кнопка входу в особистий кабінет і кнопка виходу із системи |
| Стан ПЗ після проведення випробувань: | Користувача авторизовано, дані входу збережено в cookies |

Таблиця 5.2 – Додавання алгоритму

| **Мета тесту:** | **Перевірка функції «Додавання алгоритму»** |
| --- | --- |
| Початковий стан ПЗ | Відкрита сторінка «Алгоритми» |
| Вхідні дані: | Назва алгоритму, вихідний код, опис у вільному форматі |
| Схема проведення тесту: | Натиснути кнопку «Додати алгоритм», заповнити обов’язкові поля форми коректними даними |
| Очікуваний результат: | Відкрито сторінку зі списком алгоритмів, новий алгоритм видно із загального списку алгоритмів |
| Стан ПЗ після випробувань | Алгоритм занесено в систему |

Таблиця 5.3 – Аналіз алгоритму

| **Мета тесту:** | **Перевірка функції «Аналіз алгоритму»** |
| --- | --- |
| Початковий стан ПЗ | Відкрито сторінку «Алгоритми» |
| Вхідні дані: |  |
| Схема проведення тесту: | Натиснути на посилання з назвою алгоритму |

Продовження таблиці 5.3

| **Мета тесту:** | **Перевірка функції «Аналіз алгоритму»** |
| --- | --- |
| Очікуваний результат: | Відкрито сторінку з інформацією про алгоритм та набором графіків аналітик алгоритму. Кожен графік має належну назву і підписи осей в залежності від аналітики, графіки побудовані у відповідності до тестових даних, наявних у базі даних |
| Стан ПЗ після проведення випробувань: | Дані побудованих аналітик закешовано |

Таблиця 5.4 – Побудова розкладу

| **Мета тесту:** | **Перевірка функції «Побудова розкладу для довільних вхідних даних»** |
| --- | --- |
| Початковий стан ПЗ | Відкрито сторінку «Алгоритми» |
| Вхідні дані: | Опис набору завдань і верстатів |
| Схема проведення тесту: | Натиснути кнопку «Розклад» навпроти обраного алгоритму, обрати файл з вхідними даними |
| Очікуваний результат: | Відкрито сторінку з діаграмою Ганта побудованого розкладу |
| Стан ПЗ після проведення випробувань: | Стан системи не змінено, відкрито сторінку особистого кабінету користувача |

## **Висновок до розділу**

В даному розділі систему було розглянуто з точки зору основного користувача – розробника алгоритмів, було наведено керівництво користувача.

Окрім цього, було описано випробування програмного продукту і наведено результати випробувань.

ЗАГАЛЬНІ ВИСНОВКИ

У ході виконання дипломного проекту було проаналізовано теперішній стан розвитку теорії розкладів і вивчено методи і підходи до розв’язання важковирішуваних задач календарного планування. Було детально вивчено теорію ПДС-алгоритмів, реалізовано і протестовано деякі з них.

В розділі загальних положень ми описали предметне середовище системи, що розроблюється, і навели діаграму діяльності для опису процесу діяльності і діаграму варіантів використання для опису функціональної моделі системи. Проведено огляд систем, що мають спільні властивості з даною. В описі постановки задачі було дано призначення розробки системи, цілі та задачі розробки.

Після вивчення предметного середовища було спроектовано систему підтримки розробки і аналізу ПДС-алгоритмів складання розкладів робіт на паралельних верстатах однакової продуктивності. Ми використали засоби уніфікованої мови моделювання для представлення і аналізу системи, що розроблюється, за різними критеріями.

В розділі інформаційного забезпечення було докладно описано систему з точки зору даних, що вона приймає на вхід, оброблює, зберігає і передає на вихід користувачу. Наведено фізичну діаграму бази даних системи і описано основні сутності системи і зв’язки між ними.

Розділ математичного забезпечення дав докладний опис основної математичної задачі, що вирішує система – задачі побудови оптимальних за критерієм максимізації мінімального моменту початку роботи верстатів розкладів робіт з довільними директивними термінами на паралельних верстатах однакової продуктивності. Вивчено існуючі методи вирішення даної задачі і описано новий розроблений метод – ПДС-алгоритм, проведено експериментальний аналіз його ефективності і порівняно з точним алгоритмом розв’язання даної задачі.

Описано програмне та технічне забезпечення. Для програмної реалізації спроектованої системи використовувалися засоби розробки переважно від компанії Майкрософт, такі як платформа .NET, мова програмування C#, фреймворк ASP.NET MVC, середовище розробки Visual Studio та інші. Для опису архітектури програмного забезпечення системи було наведено наступні діаграми: класів, послідовності та компонентів, дано специфікацію основних функцій.

На завершальному етапі було проведено процес тестування і відладки системи з використанням технологій модульного і інтеграційного тестування. Складено керівництво користувача і описано методику проведення випробувань, які допоможуть ввести систему в експлуатацію.

Перелік посилань

1. Згуровский М.З., Павлов А.А. Принятие решений в сетевых системах с ограниченными ресурсами: Монография.– К.: Наукова думка. – 2010. – 573 с.
2. Павлов А.А. Признаки оптимальности допустимых решений труднорешаемых задач комбина-торной оптимизации // Вісник НТУУ “КПІ”. Серія «Інформатика, управління та обчислю¬вальна техніка». – К.: “ВЕК+”, 2013. – №59 – С. 4–11.
3. Згуровский М.З. Минимизация лексикографического критерия для допустимого расписания на независимых параллельных приборах с произвольными директивными сроками [Текст] / М.З. Згуровский, А.А. Павлов, Е.Б. Мисюра // Вісник НТУУ “КПІ”. Серія «Інформатика, управління та обчислювальна техніка». – К.: “ВЕК+”, 2014. – №61. – С. 4–17.
4. Конструктивные полиномиальные алгоритмы решения индивидуальных задач из класса NP / А.А.Павлов, А.Б.Литвин, Е.Б.Мисюра, Л.А.Павлова, В.И.Родионов, под редакцией А.А.Павлова.– К.: Техника, 1993.– 126 с.
5. Pavlov A., Pavlova L. PDC-algorithms for intractable combinatorial problems. Theory and meth-odology of design.– Uzhhorod, «Karpatskij region» shelf №15, 1998.– 320 p.
6. Land, A. H. und A. G. Doig (1960). An automatic method of solving discrete programming problems. In: Econometrica 28, P. 497—520.
7. Сторінка мови програмування C# Wikipedia [Електронний ресурс], // Режим доступу: https://uk.wikipedia.org/wiki/C\_Sharp.
8. Сторінка шаблону модель-вид-контролер електронної енциклопедії Wikipedia [Електронний ресурс], // Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C-%D0%B2%D0%B8%D0%B4-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80

**Додаток А**

(Найменування програми (документа))

***Тексти програмного коду***

***Система підтримки розробки та аналізу ПДС-алгоритмів побудови розкладів робіт на паралельних верстатах однаккової продуктивності***

(Вид носія даних)

*CD-R*

(Обсяг програми (документа), арк., Мб)

*103 арк, 202 Мб*

Київ – 2017 року

**Schedule.cs**

using System;

using System.Collections.Generic;

using System.Linq;

namespace Scheduling

{

/// <summary>

/// Represents a schedule

/// </summary>

public class Schedule : List<MachineSchedule>, IComparable<Schedule>, ICloneable

{

/// <summary>

/// The criterion of optimality is satisfied

/// </summary>

public bool OptimalityCriterion { get; private set; }

#region Constructors

public Schedule()

{

}

public Schedule(IEnumerable<MachineSchedule> machineSchedules)

{

AddRange(machineSchedules);

}

#endregion

#region Implementations

public int CompareTo(Schedule second)

{

if (second == null)

{

return 1;

}

if (second.Count != Count)

{

throw new ArgumentException("The schedules must have the same number of machines.");

}

for (var i = 0; i < Count; i++)

{

if (this[i].StartTime < second[i].StartTime)

{

return -1;

}

if (this[i].StartTime > second[i].StartTime)

{

return 1;

}

}

return 0;

}

public object Clone()

{

var newSchedule = new Schedule();

foreach (var machineSchedule in this)

{

var newMachineSchedule = machineSchedule.Clone();

newSchedule.Add((MachineSchedule)newMachineSchedule);

}

newSchedule.OptimalityCriterion = OptimalityCriterion;

return newSchedule;

}

#endregion

#region Public methods

public static Schedule BuildOptimalSchedule(IEnumerable<Task> tasks, IEnumerable<Machine> machines)

{

var tasksList = tasks.ToList();

var machinesList = machines.ToList();

if (tasksList.Count == 0 || machinesList.Count == 0)

{

return new Schedule();

}

tasksList.Sort((x, y) =>

{

var res = x.Deadline.CompareTo(y.Deadline);

return res == 0 ? x.Duration.CompareTo(y.Duration) : res;

});

var m = machinesList.Count;

var n = tasksList.Count;

var queue = new List<BBNode>();

#region Build initial queue

var heuristicSchedule = BuildSchedule(tasksList, machinesList);

// First element

var startTimes = new DateTime[m];

for (var i = 0; i < m; ++i)

{

startTimes[i] = DateTime.MaxValue;

}

var target = 0;

for (var j = 0; j < heuristicSchedule.Count; ++j)

{

if (heuristicSchedule[j].Tasks.Any(x => x.Id == tasksList[n - 1].Id))

{

target = j;

break;

}

}

startTimes[target] = tasksList[n - 1].ExtremeTime;

queue.Add(new BBNode(n - 1, target, null, startTimes));

var forBranch = 0;

for (var i = n - 2; i >= 0; --i)

{

var currentTask = tasksList[i];

var node = queue[forBranch];

queue.RemoveAt(forBranch);

var nextMachine = 0;

for (var j = 0; j < heuristicSchedule.Count; ++j)

{

if (heuristicSchedule[j].Tasks.Any(x => x.Id == currentTask.Id))

{

nextMachine = j;

break;

}

}

var engaged = node.StartTimes[nextMachine] == DateTime.MaxValue;

var newPrevious = new PreviousNode

{

MachineIndex = node.MachineIndex,

Previous = node.PreviousNode

};

// Copy the array of start times

var newStartTimesForBranch = new DateTime[m];

for (var k = 0; k < m; k++)

{

newStartTimesForBranch[k] = node.StartTimes[k];

}

newStartTimesForBranch[nextMachine] = !(currentTask.Deadline > newStartTimesForBranch[nextMachine])

? currentTask.ExtremeTime

: newStartTimesForBranch[nextMachine].AddMinutes(-currentTask.Duration);

queue.Add(new BBNode(i, nextMachine, newPrevious, newStartTimesForBranch));

forBranch = queue.Count - 1;

for (var j = 0; j < m; ++j)

{

if (j != nextMachine)

{

if (node.StartTimes[j] == DateTime.MaxValue && engaged)

{

continue;

}

if (node.StartTimes[j] == DateTime.MaxValue)

{

engaged = true;

}

// Copy the array of start times

var newStartTimes = new DateTime[m];

for (var k = 0; k < m; k++)

{

newStartTimes[k] = node.StartTimes[k];

}

newStartTimes[j] = !(currentTask.Deadline > newStartTimes[j])

? currentTask.ExtremeTime

: newStartTimes[j].AddMinutes(-currentTask.Duration);

queue.Add(new BBNode(i, j, newPrevious, newStartTimes));

}

}

}

// Find record

var record = queue[forBranch];

queue.RemoveAt(forBranch);

for (var i = queue.Count - 1; i >= 0 && queue[i].TaskIndex == 0; --i)

{

var current = queue[i];

var temp = compare(record.StartTimes, current.StartTimes, m);

if (temp < 0)

{

record = current;

}

queue.RemoveAt(i);

}

// Clear the queue

for (var i = queue.Count - 1; i >= 0; --i)

{

if (compare(record.StartTimes, queue[i].StartTimes, m) < 0)

{

continue;

}

queue.RemoveAt(i);

}

#endregion

while (queue.Count != 0)

{

var current = queue[queue.Count - 1];

queue.RemoveAt(queue.Count - 1);

var newPrevious = new PreviousNode

{

MachineIndex = current.MachineIndex,

Previous = current.PreviousNode

};

var engaged = false;

var currentTaskIndex = current.TaskIndex - 1;

var currentTask = tasksList[currentTaskIndex];

for (var j = 0; j < m; ++j)

{

if (current.StartTimes[j] == DateTime.MaxValue && engaged)

{

continue;

}

if (current.StartTimes[j] == DateTime.MaxValue)

{

engaged = true;

}

// Copy the array of start times

var newStartTimes = new DateTime[m];

for (var k = 0; k < m; k++)

{

newStartTimes[k] = current.StartTimes[k];

}

newStartTimes[j] = !(currentTask.Deadline > newStartTimes[j])

? currentTask.ExtremeTime

: newStartTimes[j].AddMinutes(-currentTask.Duration);

var next = new BBNode(currentTaskIndex, j, newPrevious, newStartTimes);

if (compare(record.StartTimes, newStartTimes, m) >= 0)

{

continue;

}

// New record

if (currentTaskIndex == 0)

{

record = next;

// Clear the queue

for (var i = queue.Count - 1; i >= 0; --i)

{

if (compare(record.StartTimes, queue[i].StartTimes, m) < 0)

{

continue;

}

queue.RemoveAt(i);

}

continue;

}

queue.Add(next);

}

}

var result = new Schedule(machinesList);

result[record.MachineIndex].Tasks.AddLast(tasksList[record.TaskIndex]);

var previous = record.PreviousNode;

var index = 1;

while (previous != null)

{

result[previous.MachineIndex].Tasks.AddLast(tasksList[index]);

previous = previous.Previous;

++index;

}

for (var i = 0; i < m; i++)

{

result[i].StartTime = record.StartTimes[i];

}

result.sort();

return result;

}

#endregion

#region Service functions

private SortedSet<MachineScheduleWrapper> optimalInitialSchedulePrimary(List<Task> tasks, out int nextTaskIndex,

out bool firstCriterion)

{

nextTaskIndex = 0;

var initialAppointmentInfo = new SortedSet<MachineScheduleWrapper>();

// Engage one machine and appoint first task

var firstTask = tasks[0];

var engagedMachine = new MachineScheduleWrapper(this[0], firstTask.Deadline);

engagedMachine.Schedule.Tasks.AddLast(firstTask);

engagedMachine.Schedule.StartTime = firstTask.ExtremeTime;

initialAppointmentInfo.Add(engagedMachine);

var i = 1;

var j = 1;

firstCriterion = true;

while (j < Count && i < tasks.Count)

{

var currentTask = tasks[i];

// Check di-Ci < lp

foreach (var machineSchedule in initialAppointmentInfo)

{

var currentEndTime = machineSchedule.Schedule.StartTime;

foreach (var task in machineSchedule.Schedule.Tasks)

{

currentEndTime = currentEndTime.AddMinutes(task.Duration);

if (!((task.Deadline - currentEndTime).TotalMinutes < currentTask.Duration))

{

return null;

}

}

}

++i;

var minMachine = initialAppointmentInfo.Min;

if (minMachine.EndTime > currentTask.ExtremeTime)

{

// engage next processor

var nextMachine = new MachineScheduleWrapper(this[j], currentTask.Deadline);

nextMachine.Schedule.StartTime = currentTask.ExtremeTime;

nextMachine.Schedule.Tasks.AddLast(currentTask);

initialAppointmentInfo.Add(nextMachine);

++j;

continue;

}

initialAppointmentInfo.Remove(minMachine);

if (initialAppointmentInfo.Count > 0 && !(initialAppointmentInfo.Min.EndTime > currentTask.ExtremeTime))

{

return null;

}

minMachine.Schedule.Tasks.AddLast(currentTask);

minMachine.EndTime = minMachine.EndTime.AddMinutes(currentTask.Duration);

initialAppointmentInfo.Add(minMachine);

firstCriterion = false;

}

nextTaskIndex = i;

return initialAppointmentInfo;

}

private bool combineWithInitialPart(List<MachineScheduleWrapper> initialSchedule)

{

sort();

// Calculate delays

var delays = new List<Tuple<int, TimeSpan>>();

for (var i = 0; i < Count; i++)

{

var delay = initialSchedule[i].EndTime - this[i].StartTime;

if (delay > TimeSpan.Zero)

{

delays.Add(Tuple.Create(i, delay));

}

}

delays.Sort((x, y) => y.Item2.CompareTo(x.Item2));

// Resolve conflicts

foreach (var delayTuple in delays)

{

var index = delayTuple.Item1;

var machineSchedule = this[index];

DateTime currentStartTime;

do

{

var movingTask = machineSchedule.Tasks.First.Value;

var newIndex = -1;

var minInterval = TimeSpan.MaxValue;

var newStartTime = DateTime.MinValue;

// Find new machine

for (var i = 0; i < Count; i++)

{

var currentMachine = this[i];

var currentInitialSchedule = initialSchedule[i];

if (i == index || currentMachine.StartTime < currentInitialSchedule.EndTime)

{

continue;

}

var startTime = movingTask.Deadline < currentMachine.StartTime

? movingTask.ExtremeTime

: currentMachine.StartTime.AddMinutes(-movingTask.Duration);

if (!(newStartTime < currentInitialSchedule.EndTime))

{

var interval = currentMachine.StartTime - currentInitialSchedule.EndTime;

if (interval < minInterval)

{

newIndex = i;

minInterval = interval;

newStartTime = startTime;

}

}

}

if (newIndex < 0)

{

return false;

}

var newMachine = this[newIndex];

newMachine.StartTime = newStartTime;

newMachine.Tasks.AddFirst(movingTask);

machineSchedule.Tasks.RemoveFirst();

currentStartTime = calculateStartTime(machineSchedule.Tasks);

} while (currentStartTime < initialSchedule[index].EndTime);

}

// Merge schedules

for (var i = 0; i < Count; i++)

{

var currentMachine = this[i];

var currentInitialSchedule = initialSchedule[i];

var currentTask = currentInitialSchedule.Schedule.Tasks.Last;

while (currentTask != null)

{

currentMachine.Tasks.AddFirst(currentTask.Value);

currentTask = currentTask.Previous;

}

currentMachine.StartTime = currentInitialSchedule.Schedule.StartTime;

}

return true;

}

private static Schedule mainHeuristicAlgorithm(List<Task> remainingTasks, List<Machine> machines,

TimeSpan? adjustmentBorder = null)

{

var checkInterval = adjustmentBorder != null;

var maxInvervalValue = TimeSpan.MinValue;

if (checkInterval)

{

maxInvervalValue = (TimeSpan)adjustmentBorder;

}

var schedule = heuristicInitialAppointment(remainingTasks, machines);

remainingTasks.Sort((x, y) =>

{

var t = y.Deadline.CompareTo(x.Deadline);

return t == 0 ? x.Duration.CompareTo(y.Duration) : t;

});

while (remainingTasks.Count != 0)

{

// Select feasible tasks for the current iteration

var feasibleTasks = new List<Task>();

foreach (var task in remainingTasks)

{

var feasible = false;

for (var j = 0; j < machines.Count; j++)

{

if (schedule[j].StartTime > task.Deadline)

{

continue;

}

feasible = true;

break;

}

if (feasible)

{

feasibleTasks.Add(task);

}

else

{

break;

}

}

feasibleTasks.Sort((x, y) =>

{

var t = x.Duration.CompareTo(y.Duration);

return t == 0 ? y.Deadline.CompareTo(x.Deadline) : t;

});

var remMachines = new List<MachineSchedule>(schedule);

while (remMachines.Count != 0)

{

if (feasibleTasks.Count == 0)

{

break;

}

var currentTask = feasibleTasks[0];

feasibleTasks.Remove(currentTask);

MachineSchedule best = null;

foreach (var machine in remMachines)

{

if (machine.StartTime > currentTask.Deadline)

{

continue;

}

if (best == null || best.StartTime < machine.StartTime)

{

best = machine;

}

}

if (best != null)

{

best.Tasks.AddFirst(currentTask);

best.StartTime = best.StartTime.AddMinutes(-currentTask.Duration);

remMachines.Remove(best);

remainingTasks.Remove(currentTask);

}

}

foreach (var machine in remMachines)

{

if (remainingTasks.Count == 0)

{

break;

}

var currentTask = remainingTasks[0];

machine.Tasks.AddFirst(currentTask);

machine.StartTime = currentTask.ExtremeTime;

remainingTasks.Remove(currentTask);

}

// Execute adjustment algorithm if needed

if (checkInterval)

{

var maxStartTime = DateTime.MinValue;

var minStartTime = DateTime.MaxValue;

foreach (var machine in schedule)

{

if (machine.StartTime > maxStartTime)

{

maxStartTime = machine.StartTime;

}

if (machine.StartTime < minStartTime)

{

minStartTime = machine.StartTime;

}

}

if (maxStartTime - minStartTime > maxInvervalValue)

{

schedule.startTimesAdjustment();

}

}

}

schedule.startTimesAdjustment();

return schedule;

}

private static Schedule secondHeuristicAlgorithm(List<Task> remainingTasks, List<Machine> machines)

{

var schedule = heuristicInitialAppointment(remainingTasks, machines);

remainingTasks.Sort((x, y) =>

{

var t = y.Deadline.CompareTo(x.Deadline);

return t == 0 ? x.Duration.CompareTo(y.Duration) : t;

});

var sortedSet = new SortedSet<MachineSchedule>(schedule);

while (remainingTasks.Count != 0)

{

var lastMachine = sortedSet.Max;

// Searching for a feasible task with minimal duration

Task best = null;

DateTime startTime;

foreach (var task in remainingTasks)

{

if (lastMachine.StartTime > task.Deadline)

{

break;

}

if (best == null || task.Duration < best.Duration)

{

best = task;

}

}

if (best == null)

{

foreach (var task in remainingTasks)

{

if (best == null || task.ExtremeTime > best.ExtremeTime)

{

best = task;

}

}

startTime = best.ExtremeTime;

}

else

{

startTime = lastMachine.StartTime.AddMinutes(-best.Duration);

}

sortedSet.Remove(lastMachine);

lastMachine.Tasks.AddFirst(best);

lastMachine.StartTime = startTime;

sortedSet.Add(lastMachine);

remainingTasks.Remove(best);

}

schedule.startTimesAdjustment();

return schedule;

}

private static Schedule thirdHeuristicAlgorithm(List<Task> tasks, List<Machine> machines)

{

var schedule = new Schedule(machines);

var machineSchedules = new SortedSet<MachineScheduleWrapper>();

// Engage one machine and appoint first task

var firstTask = tasks[0];

var engagedMachine = new MachineScheduleWrapper(schedule[0], firstTask.Deadline);

engagedMachine.Schedule.Tasks.AddLast(firstTask);

engagedMachine.Schedule.StartTime = firstTask.ExtremeTime;

machineSchedules.Add(engagedMachine);

var j = 1;

for (var i = 1; i < tasks.Count; i++)

{

var currentTask = tasks[i];

var minMachine = machineSchedules.Min;

MachineScheduleWrapper currentBest = null;

if (minMachine.EndTime > currentTask.ExtremeTime)

{

// Find best machine (engage next if needed)

if (j < schedule.Count)

{

// Engage next machine

currentBest = new MachineScheduleWrapper(schedule[j], currentTask.Deadline);

currentBest.Schedule.StartTime = currentTask.ExtremeTime;

j++;

}

else

{

// Find best (with offset)

DateTime[] bestStartTimes = null;

var k = 0;

var bestOffset = TimeSpan.MinValue;

foreach (var machineSchedule in machineSchedules)

{

var startTimes = machineSchedules.Select(m => m.Schedule.StartTime).ToArray();

var offset = currentTask.ExtremeTime - machineSchedule.EndTime;

startTimes[k] = startTimes[k].Add(offset);

if (bestStartTimes == null || compare(bestStartTimes, startTimes, schedule.Count) < 0)

{

currentBest = machineSchedule;

bestStartTimes = startTimes;

bestOffset = offset;

}

k++;

}

machineSchedules.Remove(currentBest);

currentBest.Schedule.StartTime = currentBest.Schedule.StartTime.Add(bestOffset);

currentBest.EndTime = currentTask.Deadline;

}

}

else

{

// The task is feasible for the current best machine

currentBest = minMachine;

machineSchedules.Remove(currentBest);

currentBest.EndTime = currentBest.EndTime.AddMinutes(currentTask.Duration);

}

currentBest.Schedule.Tasks.AddLast(currentTask);

machineSchedules.Add(currentBest);

}

schedule.startTimesAdjustment();

return schedule;

}

private static Schedule heuristicInitialAppointment(List<Task> tasks, List<Machine> machines)

{

var schedule = new Schedule(machines);

foreach (var machineSchedule in schedule)

{

if (tasks.Count == 0)

{

break;

}

var task = tasks[tasks.Count - 1]; // Last task

machineSchedule.Tasks.AddLast(task);

machineSchedule.StartTime = task.ExtremeTime;

tasks.Remove(task);

}

return schedule;

}

private void startTimesAdjustment()

{

var exit = false;

while (!exit)

{

sort();

exit = true;

for (var i = 0; i < Count - 1; i++)

{

var currentMachine = this[i];

if (currentMachine.Tasks.Count == 0)

{

continue;

}

var currentStartTime = currentMachine.StartTime;

while (shiftCondition(currentMachine.Tasks))

{

var currentTaskNode = currentMachine.Tasks.First;

var shiftingTask = currentTaskNode.Value;

var currentTaskEnd = currentStartTime.AddMinutes(shiftingTask.Duration);

var shifted = false;

while (currentTaskNode.Next != null &&

!(currentTaskEnd.AddMinutes(currentTaskNode.Next.Value.Duration) > shiftingTask.Deadline))

{

currentTaskNode = currentTaskNode.Next;

currentTaskEnd = currentTaskEnd.AddMinutes(currentTaskNode.Value.Duration);

shifted = true;

}

if (!shifted)

{

break;

}

currentMachine.Tasks.AddAfter(currentTaskNode, shiftingTask);

currentMachine.Tasks.RemoveFirst();

}

var movingTask = currentMachine.Tasks.First.Value;

MachineSchedule targetMachine = null;

for (var j = Count - 1; j > i; j--)

{

if (targetMachine != null && this[j].StartTime < movingTask.Deadline)

{

break;

}

targetMachine = this[j];

}

if (targetMachine != null)

{

var newStartTime = (movingTask.Deadline < targetMachine.StartTime)

? movingTask.ExtremeTime

: targetMachine.StartTime.AddMinutes(-movingTask.Duration);

if (newStartTime > currentStartTime)

{

currentMachine.Tasks.RemoveFirst();

currentMachine.StartTime = calculateStartTime(currentMachine.Tasks);

targetMachine.Tasks.AddFirst(movingTask);

targetMachine.StartTime = newStartTime;

exit = false;

break;

}

}

}

}

}

private static bool shiftCondition(LinkedList<Task> tasks)

{

var first = tasks.First;

var second = first.Next;

return second != null && first.Value.Duration > second.Value.Duration;

}

private static DateTime calculateStartTime(LinkedList<Task> tasks)

{

if (tasks.Count == 0)

{

return DateTime.MaxValue;

}

var currentTaskNode = tasks.Last;

var currentStartTime = currentTaskNode.Value.ExtremeTime;

while (currentTaskNode.Previous != null)

{

currentTaskNode = currentTaskNode.Previous;

currentStartTime = (currentTaskNode.Value.Deadline < currentStartTime)

? currentTaskNode.Value.ExtremeTime

: currentStartTime.AddMinutes(-currentTaskNode.Value.Duration);

}

return currentStartTime;

}

private static readonly List<int> \_firstComparedIndexes = new List<int>();

private static readonly List<int> \_secondComparedIndexes = new List<int>();

private static int compare(DateTime[] first, DateTime[] second, int count)

{

\_firstComparedIndexes.Clear();

\_secondComparedIndexes.Clear();

for (var i = 0; i < count; i++)

{

var firstMinTimeIndex = -1;

var firstMinTime = DateTime.MaxValue;

for (var j = 0; j < count; j++)

{

var current = first[j];

if ((firstMinTimeIndex < 0 || current < firstMinTime) && !\_firstComparedIndexes.Contains(j))

{

firstMinTimeIndex = j;

firstMinTime = current;

}

}

var secondMinTimeIndex = -1;

var secondMinTime = DateTime.MaxValue;

for (var j = 0; j < count; j++)

{

var current = second[j];

if ((secondMinTimeIndex < 0 || current < secondMinTime) && !\_secondComparedIndexes.Contains(j))

{

secondMinTimeIndex = j;

secondMinTime = current;

}

}

if (firstMinTime < secondMinTime)

{

return -1;

}

if (firstMinTime > secondMinTime)

{

return 1;

}

\_firstComparedIndexes.Add(firstMinTimeIndex);

\_secondComparedIndexes.Add(secondMinTimeIndex);

}

return 0;

}

private void sort()

{

Sort(compareMachineSchedules);

}

private int compareMachineSchedules(MachineSchedule x, MachineSchedule y)

{

var stCompRes = x.StartTime.CompareTo(y.StartTime);

return stCompRes == 0 ? x.Machine.Id.CompareTo(y.Machine.Id) : stCompRes;

}

#endregion

#region Service types

private class MachineScheduleWrapper : IComparable<MachineScheduleWrapper>, ICloneable

{

public MachineSchedule Schedule { get; }

public DateTime EndTime { get; set; }

public MachineScheduleWrapper(MachineSchedule schedule) : this(schedule, DateTime.MinValue)

{

}

public MachineScheduleWrapper(MachineSchedule schedule, DateTime endTime)

{

Schedule = schedule;

EndTime = endTime;

}

public int CompareTo(MachineScheduleWrapper other)

{

var endTimeCompResult = EndTime.CompareTo(other.EndTime);

return endTimeCompResult == 0

? Schedule.Machine.Id.CompareTo(other.Schedule.Machine.Id)

: endTimeCompResult;

}

public object Clone()

{

return new MachineScheduleWrapper((MachineSchedule)Schedule.Clone(), EndTime);

}

}

private class SuspectedTask : IComparable<SuspectedTask>

{

public Task Task { get; }

public List<int> PossibleMachines { get; }

public SuspectedTask(Task task, List<int> possibleMachines)

{

Task = task;

PossibleMachines = possibleMachines;

}

public int CompareTo(SuspectedTask other)

{

var dealineCompResult = Task.Deadline.CompareTo(other.Task.Deadline);

return dealineCompResult == 0

? Task.Id.CompareTo(other.Task.Id)

: dealineCompResult;

}

}

private class BBNode

{

public int TaskIndex { get; }

public int MachineIndex { get; }

public PreviousNode PreviousNode { get; }

public DateTime[] StartTimes { get; }

public BBNode(int taskIndex, int machineIndex, PreviousNode previous, DateTime[] startTimes)

{

TaskIndex = taskIndex;

MachineIndex = machineIndex;

PreviousNode = previous;

StartTimes = startTimes;

}

}

private class PreviousNode

{

public int MachineIndex { get; set; }

public PreviousNode Previous { get; set; }

}

#endregion

}

}