

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Курсова робота з освітнього компоненту

«Моделювання систем. Курсова робота»

Тема: Універсальний алгоритм імітації для мереж масового обслуговування

|  |  |
| --- | --- |
| **Керівник**:  Стеценко І. В.  «Допущено до захисту»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2023 р.  Захищено з оцінкою  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Члени комісії:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Виконавець**:  Зусько Владислав Юрійович  студент групи ІП-02  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «18» грудня 2023 р.  Інна СТЕЦЕНКО |

**Київ – 2023**

## Анотація

Курсова робота: 60 сторінок, 4 рисунки, 8 таблиць, 1 додаток, 3 джерела літератури.

Об'єкт дослідження: Замкнена інформаційна система, яка обробляє різні види задач з використанням оперативної пам'яті ЕОМ.

Мета роботи: Визначення впливу різних вхідних параметрів на завантаженість та розмір черги елементів системи.

Метод дослідження: Імітаційне моделювання інформаційної системи. Проведено дослідження та розроблено програмну реалізацію інформаційної моделі. Проведено експерименти з різними вхідними параметрами. Проаналізовано результати для визначення впливу різних показників на очікувані характеристики.

Результати:

Основні конструктивні, технологічні характеристики і показники: Розподіл завдань за експоненціальним законом, можливість паралельного виконання завдань різних класів.

Значимість роботи і висновки: Розроблена модель дозволяє ефективно вивчати вплив різних параметрів на роботу інформаційної системи. Визначено середню кількість вантажівок біля екскаваторів та дробівки, коефіцієнт завантаження екскаваторів та дробівки.

Прогнози і припущення щодо розвитку об'єкта дослідження: Подальші дослідження можуть включати в себе розгляд додаткових дисциплін обслуговування та кількості обслуговуючих пристроїв в моделі.

ІМІТАЦІЙНА МОДЕЛЬ, МОДЕЛЮВАННЯ СИСТЕМ, ІНФОРМАЦІЙНА СИСТЕМА, ЗАМКНЕНА СИСТЕМА.

## Зміст

[Перелік умовних позначень 5](#_Toc153823281)

[Вступ 6](#_Toc153823282)

[Постановка задачі 7](#_Toc153823283)

[1 Концептуальна модель 9](#_Toc153823284)

[2 Формалізована модель 12](#_Toc153823285)

[2.1 Вибір формалізму 12](#_Toc153823286)

[2.2 Перенесення формалізму 12](#_Toc153823287)

[3 Реалізація моделі 15](#_Toc153823288)

[3.1 Структура моделі 15](#_Toc153823289)

[3.2 Опис принципу роботи моделі 16](#_Toc153823290)

[3.3 Верифікація 18](#_Toc153823291)

[4 Проведення експериментів 19](#_Toc153823292)

[4.1 Проведення факторного експерименту 19](#_Toc153823293)

[4.2 Тактичне планування 19](#_Toc153823294)

[4.3 Стратегічне планування 21](#_Toc153823295)

[5 Інтерпретація результатів моделювання 27](#_Toc153823296)

[5.1 Інтерпретація отриманих результатів 27](#_Toc153823297)

[5.2 Пропозиції щодо удосконалення системи 27](#_Toc153823298)

[Висновок 29](#_Toc153823299)

[Список використаних джерел 30](#_Toc153823300)

[Додатки 31](#_Toc153823301)

[Додаток А. Лістинг програмного коду системи 31](#_Toc153823302)

## Перелік умовних позначень

|  |  |
| --- | --- |
| FIFO | – First in First out, в обчислювальній техніці та теорії систем метод "першим прийшов – першим вийшов", – це метод організації маніпуляцій зі структурою даних, при якому найстаріший запис, або "голова" черги, обробляється в першу чергу. |
| СМО | – система масового обслуговування. |

## Вступ

У сучасних реаліях, коли ефективне використання ресурсів стає пріоритетним завданням, важливість дослідження ефективності різних систем масового обслуговування є особливо високою. Ця необхідність виникає в найрізноманітніших галузях, починаючи від банківських відділень і закінчуючи центрами обробки даних, охоплюючи транспортні системи та будівельні майданчики.

Одним з ключових методів дослідження ефективності є моделювання таких систем. За наявності відповідних даних для моделювання можна оцінити різноманітні показники, такі як частота відмов, завантаження вузлів обслуговування, середня довжина черги та інші. Аналізуючи ці дані, можна здійснити прогноз кількості виконаних завдань, передбачити відмови системи та визначити середні очікування обслуговування. Важливим аспектом є можливість оптимізації поточних ресурсів чи зміни стратегії обслуговування для оцінки її переваг та недоліків порівняно з іншими стратегіями. Все це стає невід'ємною частиною оцінки ефективності моделі обслуговування для бізнесу, дозволяючи не лише зрозуміти її поточний стан, а й вносити необхідні зміни для оптимізації.

Метою нашої курсової роботи є розробка імітаційного алгоритму для замкненої інформаційної системи, спроможної обробляти різні види задач. Наше дослідження передбачає аналіз роботи системи для визначення завантаженості та розміру черги її елементів.

## Постановка задачі

У кар’єрі вантажівки доставляють руду від трьох екскаваторів до одної дробівки. Вантажівки приписані до певних екскаваторів, так що кожна вантажівка завжди повертається до свого екскаватора після того, як вивантажила руду біля дробівки. Використовуються вантажівки двох видів; вантажопідіймальністю 20 і 50 тон. Вантажопідіймальність впливає на час вантаження машин екскаватором, час переїзду до дробівки, час розвантаження та час повернення до екскаватору. Числові характеристики цих величин наведені у таблиці.

Таблиця – Часові затримки

|  |  |  |
| --- | --- | --- |
| **Процес** | **Час для 20-тонної вантажівки** | **Час для 50-тонної вантажівки** |
| **Вантаження** | Випадкова величина з експоненціальним розподілом із середнім значенням 5 хвилин | Випадкова величина з експоненціальним розподілом із середнім значенням 10 хвилин |
| **Переїзду до дробівки** | Постійна величина 2,5 хвилини | Постійна величина 3 хвилини |
| **Розвантаження** | Випадкова величина з експоненціальним розподілом із середнім значенням 5 хвилин | Випадкова величина з експоненціальним розподілом із середнім значенням 4 хвилини |
| **Повернення до екскаватору** | Постійна величина 1,5 хвилини | Постійна величина 2 хвилини |

До кожного екскаватору закріплені дві вантажівки 20-тонна та 50-тонна. Всі черги до екскаваторів мають дисципліну обслуговування FIFO. У черзі до дробівки машини стоять у порядку зменшення їх вантажопідіймальності, а при однаковій вантажопідіймальності – за правилом FIFO.

Метою моделювання є визначення:

* середньої кількості вантажівок у кожного екскаватора та у дробівки;
* коефіцієнти завантаження кожного екскаватора та дробівки.

## 1 Концептуальна модель

Сформуємо концептуальну модель, що відображає принцип роботи системи. Виокремимо етапи та елементи її роботи.



Рисунок 2.1 – Концептуальна модель системи

У склад моделі входять:

* надходження вантажівки;
* черга вантаження;
* процес вантаження;
* проїзд до дробівки;
* черга дробіння;
* процес дробіння;
* повернення.

Порядок просування вантажівки (задачі):

* приїзд до ескаватора;
* пересування до дробівки;
* переїзд назад.

Вхідними параметрами виступають:

* вантажівки різної вантажопідйомності;
* дисципліна процесів.

Встановлені обмеження:

* вантажівки можуть бути 2-ох видів, з вантажопідйомністю 20 та 50 тон;
* до кожного екскаватору закріплені дві вантажівки 20-тонна та 50-тонна;
* усі черги до екскаваторів формуються за принципом FIFO;
* у черзі до дробівки машини стоять у порядку зменшення їх вантажопідіймальності, а при однаковій вантажопідіймальності – за правилом FIFO.

Опишемо надану концептуальну модель. Перш за все, вантажівки надходять до екскаваторів. Аби екскаватор навантажив вантажівку, вона має дочекатись своєї черги. Далі, вантажівка слідує до дробівки. Вона теж має свою чергу. Після успішного виконання дробіння, вантажівка повертається назад до екскаватора. Бачимо, що система має обмежену і фіксовану кількість задач, що рухаються у циклі. Вхідними параметрами слугують час навантаження різних автомобілів, час переїзду та розвантаження, час повернення. Вихідними параметрами є середня кількість вантажівок у екскаваторів та дробівки і їхні коефіцієнти завантаженості.

## 2 Формалізована модель

### 2.1 Вибір формалізму

Для надання формалізованого вигляду системі було обрано формалізм мереж масового обслуговування. Він є оптимальним рішенням з ряду причин. Ця методика добре лягає на системи, які є подійними та метою яких є обробка різних створених задач. Також формалізм містить різноманітні сутності, що дозволяють як моделювати черги, так і переходи між різними етапами системи, вказуючи час кожного з них. Додатково можна вказати, що СМО є дуже корисними для оцінок продуктивності як системи загалом, так і її частин окремо.

Таким чином, обраний формалізм підхід дозволить нам точно змоделювати умови вказаної задачі та коректно оцінити потрібні вихідні метрики. Моделі такого типу можуть включати в себе наступні частини:

* дуги, що слугують для руху в межах системи/систем;
* черги, що відображають накопичення необроблених елементів;
* канали обслуговування, що відповідають за обробку задач;
* розгалуження, що відображають можливість альтернативних шляхів руху задачі;
* блокування, які надають змогу тимчасово припинити надходження задач за певною умовою;
* багатоканальну СМО, що містять декілька каналів.

### 2.2 Перенесення формалізму

Перенесемо умову задачі на елементи формалізму Сформуємо схему, виокремивши частини системи. Наведемо сконструйовану модель.

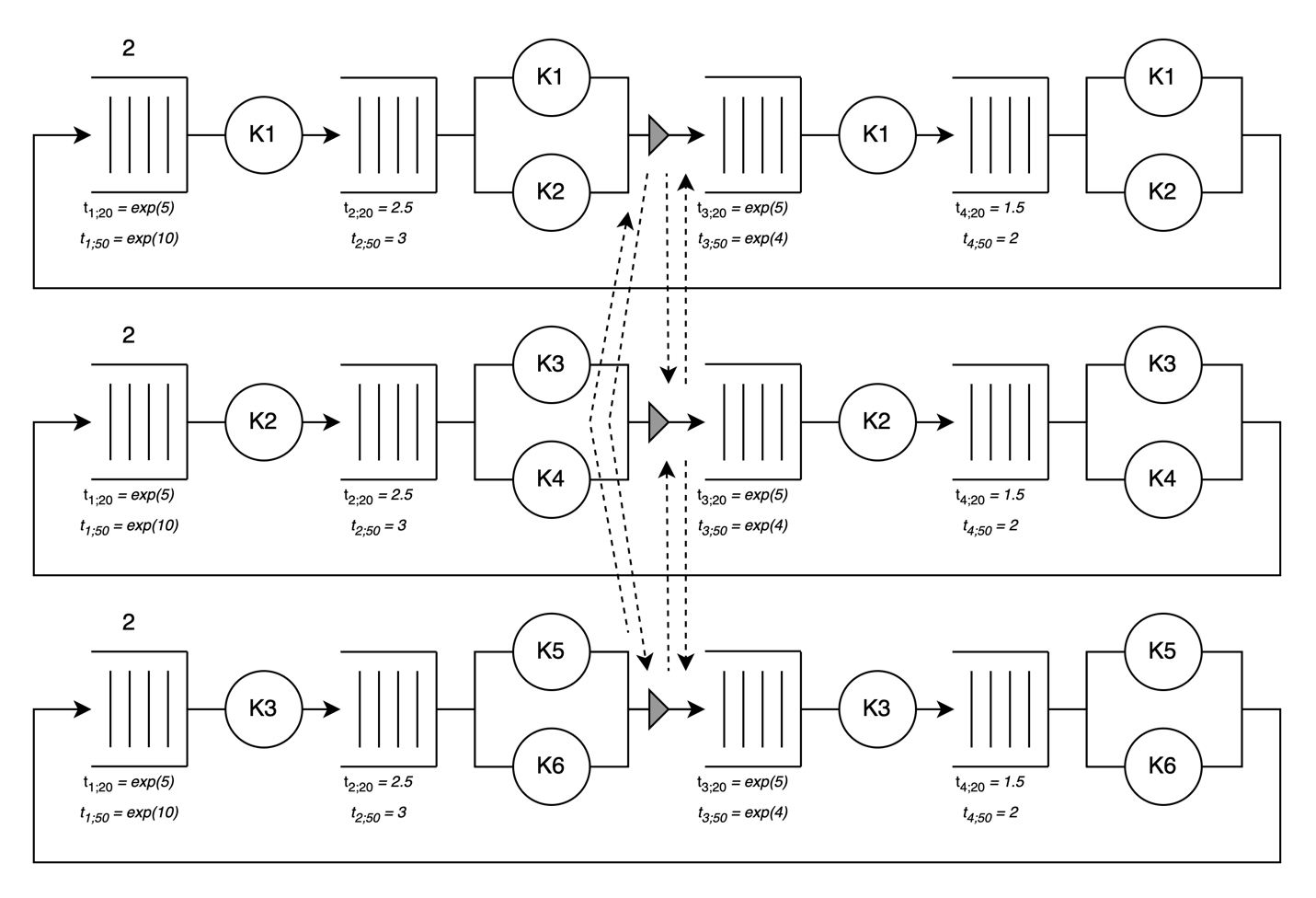


Рисунок 2.1 – Формалізована модель

Наведемо список вхідних параметрів моделі:

* – час вантаження вантажівки вагою 20 тон;
* – час вантаження машини вагою 50 тон;
* – час переїзду транспорту вагою 20 тон до дробівки;
* – час переїзду транспорту вагою 50 тон до дробівки;
* – час розвантаження 20-ти тонної вантажівки;
* – час розвантаження 50-ти тонної вантажівки;
* – час повернення вантажівки на 20 тон назад до екскаватора;
* – час повернення вантажівки на 50 тон назад до екскаватора.

Вихідні параметри:

* – середня кількість вантажівок у 1-го екскаватора;
* – середня кількість вантажівок у 2-го екскаватора;
* – середня кількість вантажівок у 3-го екскаватора;
* – середня кількість вантажівок у дробівки;
* – коефіцієнт завантаження 1-го екскаватора;
* – коефіцієнт завантаження 2-го екскаватора;
* – коефіцієнт завантаження 3-го екскаватора;
* – коефіцієнт завантаження дробівки.

Просування часу відбуватиметься за принципом .

Метою виконання задачі є оцінка середньої кількості вантажівок у кожного екскаватора та дробівки. Для цього можемо використати наступну формулу:

(2.1)

,

де – розмір черги в поточний момент часу елементу , -  різниця між поточним часом елемента та наступною точкою часу моделі, – кількість завдань, що були виконані елементом, – час моделювання.

Також, в ході моделювання ми повинні визначити коефіцієнт завантаження кожного екскаватора та дробівки. Скористаймось формулою:

(2.2)

,

де – час роботи -того елемента для виконання операцій над -тим об’єктом, -  різниця між поточним часом елемента та наступною точкою часу моделі, – кількість завдань, що були виконані елементом, – час моделювання.

## 3 Реалізація моделі

### 3.1 Структура моделі

Опишемо структур та класи для реалізації імітаційної моделі. Наведемо основні доменні класи:

Таблиця 3.1 – Опис основних класів

|  |  |
| --- | --- |
| **Клас** | **Опис** |
| **Model** | Основний клас моделі системи, що оркеструє запуск потрібних процесів і просування часу |
| **Element** | Базовий абстрактний клас елементу, що містить основні властивості |
| **Worker** | Об’єкт, що відповідає за виконання задач, що надійшли, в межах елементу системи |
| **ElementWithResource** | Базовий елемент, що потребує для обробки певного ресурсу |
| **ElementWithLimitedResource** | Елемент, що має має обмежену кількість ресурсу (каналів) для обробки задач |
| **ElementWithUnlimitedResource** | Елемент, що має безмежний ресурс і не потребує черги |

Додамо основні допоміжні конструкції:

Таблиця 3.2 – Опис допоміжних класів

|  |  |
| --- | --- |
| **Клас** | **Опис** |
| **Queue** | Реалізація стандартної черги |
| **PriorityQueue** | Реалізація черги з пріоритетом за певним компаратором |
| **Next** | Інтефейс, що реалізовує вибір наступного елементу для переходу. Має багато реалізацій |
| **Random** | Клас генерації випадкових чисел, використовуючи різні стратегії |
| **DelayGenerator** | Генератор затримок, спираючись на різні види формування випадкових чисел |
| **Settings** | Службові налаштування ситеми |

### 3.2 Опис принципу роботи моделі

Роглянемо клас Model. Його метою є реалізація маркування входу задач до процесів у правильному порядку та просування часу. Основним його методом є simulate. Перш за все, він обирає найближчу в часі подію, проходячись по усіх елементах системи. Далі він здійснює перевірку наявності елемента та калькуляцію базової статистики на знайденому елементі. Опісля здійснюється просування моделі в часі, та оновлення поточного часу інших елементів. У підсумку викликається метод на позначення виходу з елемента для всіх таких, що закінчили свою роботу на даний оновлений момент часу. Також, клас забезпечує обрахунок підсумкової статистики.

Перейдемо до базового класу Element. Він містить опис ключових опцій елементу, а саме: ідентифікатора, переходу, поточного та наступного часу та статистичні поля, такі як: кількість входів задач, сума загального часу між входженнями та виходами задачі. Також, клас забезпечує наявність обрахунку вказаних статистичних даних та наявність базових методів взаємодії.

Розберемо детальніше класи ElementWith\*Resource. Класом-робітником для них виступає Worker. Він допомагає працювати каналам, кожен з яких має свій час до звільнення, стан та об’єкт, що знаходиться на обробці в ньому. Відповідно ElementWithResource визначає такий елемент, для функціонування якого необхідна певна кількість каналів. Спираючи на це, він реалізовує базові операції над каналами.

Цей клас має 2 кінцеві класи, зокрема: ElementWithLimitedResource та ElementWithUnlimitedResource.

Розглянемо 1-ий з них. Його метою є реалізація такого елемента, який має певну чергу задач та обмежену кількість каналів, що працюють одночасно. Кількість таких каналів визначається на етапі конструювання класу. З використанням методу входу inAct та методу виходу outAct він формує підбір, використання, звільнення та займання наявних каналів. Також, вказані методи контролюють наповнення черги. Додатково, клас зберігає показник розміру черги протягом часу моделювання.

Дещо протилежним за дією є клас ElementWithUnlimitedResource. Він допомагає прибрати непотрібну тут чергу та встановлення кількості каналів, додавши змогу передати функцію створення нових каналів за потреби. Відповідно клас функціонує таким чином, що за умови надходження задачі або створюється новий канал (якщо всі існуючі зайняті), або перевикористовуються існуючі. Таким чином ми не потребуємо черги та якихось обмежень щодо кількості каналів обробки.

Важливим для просування системи є інтерфейс Next. Він потрібен для можливості переходів між елементами незалежно від природи та причин переходу. Таким чином, на етапі конструювання загальної моделі, ми можемо встановити елементам необхідну логіку переходу, адже інтерфейс забезпечує поліморфізм реалізацій. Наша система містить наступні класи: SingleNext, RandomNext, PrioritizedNext, ProbabilisticNext, ConditionalNext. Кожен з класів реалізує метод, що вимагається інтерфейсом – getNextElement, проте приймає різні за структурою значення у конструкторі для гнучкого налаштування переходу.

Здійснимо верифікацію моделі, почергово змінюючи властивості моделі і нотуючи результати.

### 3.3 Верифікація

Здійснимо верифікацію моделі, почергово змінюючи вхідні параметри моделі та нотуючи вихідні значення.

Таблиця 3.3 – Верифікація моделі

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Прогон** | **Вхідні змінні** | | | | | | | | **Вихідні змінні** | | | | | | | |
| **t120** | **t150** | **t220** | **t250** | **t320** | **t350** | **t420** | **t450** | **Q1** | **Q2** | **Q3** | **Q4** | **M1** | **M2** | **M3** | **M4** |
| **1** | exp(5) | exp(10) | 2.5 | 3 | exp(5) | exp(4) | 1.5 | 2 | 0.22 | 0.26 | 0.18 | 1.69 | 0.59 | 0.61 | 0.60 | 0.90 |
| **2** | exp(7) | exp(10) | 2.5 | 3 | exp(5) | exp(4) | 1.5 | 2 | 0.19 | 0.15 | 0.18 | 2.10 | 0.51 | 0.58 | 0.55 | 0.93 |
| **3** | exp(5) | exp(15) | 2.5 | 3 | exp(5) | exp(4) | 1.5 | 2 | 0.33 | 0.22 | 0.32 | 1.54 | 0.65 | 0.61 | 0.68 | 0.84 |
| **4** | exp(5) | exp(15) | 5 | 3 | exp(5) | exp(4) | 1.5 | 2 | 0.27 | 0.12 | 0.21 | 1.54 | 0.65 | 0.56 | 0.56 | 0.92 |
| **5** | exp(5) | exp(10) | 2.5 | 5 | exp(5) | exp(4) | 1.5 | 2 | 0.14 | 0.18 | 0.18 | 1.63 | 0.51 | 0.54 | 0.52 | 0.92 |
| **6** | exp(5) | exp(10) | 2.5 | 3 | exp(10) | exp(4) | 1.5 | 2 | 0.13 | 0.11 | 0.11 | 2.55 | 0.54 | 0.47 | 0.50 | 0.94 |
| **7** | exp(5) | exp(10) | 2.5 | 3 | exp(5) | exp(6) | 1.5 | 2 | 0.14 | 0.08 | 0.13 | 2.52 | 0.41 | 0.48 | 0.48 | 0.97 |
| **8** | exp(5) | exp(10) | 2.5 | 3 | exp(5) | exp(4) | 3 | 2 | 0.15 | 0.19 | 0.17 | 1.72 | 0.58 | 0.65 | 0.55 | 0.92 |
| **9** | exp(5) | exp(10) | 2.5 | 3 | exp(5) | exp(4) | 1.5 | 4 | 0.13 | 0.18 | 0.19 | 2.02 | 0.44 | 0.48 | 0.56 | 0.91 |

Спираючись на отримані значення, ми можемо висловити деякі спостереження. Бачимо, що при збільшенні тривалості обробки, збільшувалась також черга коло відповідних екскаваторів. Додатково спостерігаємо збільшення частки робочого часу таких екскаваторів. Також, можемо зазначити, що при збільшенні часу обробки на етапі дробівки, біля неї дуже значно зростає черга та, звісно, час, витрачений на обслуговування, теж дуже піднявся. Оцінюючи підсумково таблицю верифікації, впевнюємось, що моделювання процесу проходить правильно.

## 4 Проведення експериментів

### 4.1 Проведення факторного експерименту

Перш за все визначимо, що факторний експеримент є одним з методів дослідження певного набору факторів на деякі вихідні показники моделі.

Для нашої задачі скористаємось 3 факторами, а саме: - “час переїзду 20-ти тонної вантажівки”, - “час переїзду 50-ти тонної вантажівки”, - “час повернення 50-ти тонної вантажівки”. Область змінюваних факторів є . Як відгук моделі обрано “середня кількість вантажівок у дробівки”. Також, припустимо лінійну залежність від факторів, тож кількість рівнів буде рівною 2. Точність встановимо як , а довірчу ймовірність - 0.95.

### 4.2 Тактичне планування

Метою планування постає визначення 2 опцій: тривалості прогону та кількість прогонів загалом.

Для розрахування кількості прогонів скористаємось нерівністю Чебишева, підставивши необхідні значення:

(4.1)

.

Згідно з розрахунками ми визначили, що кількість необхідних прогонів моделі становить 20.

Тепер, варто визначити тривалість перехідного періоду, аби в подальшому будувати правильні розрахунки. Для цього проведемо 4 запуски моделі з часом моделювання рівним 1000 часових одиниць. Заміри цільового показника будемо здійснювати кожних 50 одиниць часу.

Таблиця 4.1 – Результати пробних прогонів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Час** | **Прогін** | | | |
| **1** | **2** | **3** | **4** |
| **50** | 1.10 | 0.79 | 2.49 | 0.84 |
| **100** | 2.30 | 1.71 | 2.51 | 0.96 |
| **150** | 1.62 | 1.86 | 3.00 | 1.70 |
| **200** | 1.46 | 1.86 | 2.48 | 1.85 |
| **250** | 1.33 | 1.86 | 2.06 | 1.94 |
| **300** | 1.17 | 1.97 | 1.91 | 2.20 |
| **350** | 1.17 | 1.99 | 1.74 | 2.26 |
| **400** | 1.25 | 1.82 | 1.82 | 2.23 |
| **450** | 1.30 | 1.83 | 1.92 | 2.27 |
| **500** | 1.45 | 1.80 | 1.97 | 2.10 |
| **550** | 1.50 | 1.86 | 1.93 | 1.99 |
| **600** | 1.66 | 1.98 | 2.05 | 2.08 |
| **650** | 1.56 | 2.09 | 2.06 | 2.14 |
| **700** | 1.54 | 1.98 | 1.96 | 2.12 |
| **750** | 1.69 | 2.05 | 2.02 | 2.14 |
| **800** | 1.76 | 2.07 | 1.99 | 2.18 |
| **850** | 1.82 | 2.07 | 2.08 | 2.23 |
| **900** | 1.85 | 2.03 | 2.04 | 2.20 |
| **950** | 1.87 | 2.09 | 2.13 | 2.17 |
| **1000** | 1.92 | 2.12 | 2.17 | 2.22 |

Рисунок 4.1 – Графік залежності показника від часу

З рисунку бачимо, що перехідний період містить орієнтовну тривалість в 900 одиниць часу.

### 4.3 Стратегічне планування

Метою цього планування є оцінка впливу потрібних факторів на відгук моделі. Так як ми плануємо здійснити кількісну оцінку, скористаємось методом регресійного аналізу.

Перш за все, розрахуймо значення факторів. Спираючись на проміжок, який ми визначили для них, отримуємо наступну таблицю.

Таблиця 4.2 – Значення факторів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Фактор** | **Ximin** | **Ximax** | **Xi0** | **Δi** |
| **x1** | 0.60 | 4.40 | 2.50 | 1.90 |
| **x2** | 1.10 | 4.90 | 3.00 | 1.90 |
| **x3** | 0.10 | 3.90 | 2.00 | 1.90 |

Здійснимо прогони моделі з потрібними значеннями факторів протягом 2000 часових одиниць, вважаючи 900 з них перехідним періодом. Сформуємо код для запуску моделі згідно з потрібними параметрами.

*import* { *ExponentialDelayGenerator*, *ConstantDelayGenerator* } *from* './utils';

*import* { *ConditionalNext*, *SingleNext* } *from* './elements/helpers';

*import* {

*Worker*,

*ElementWithLimitedResource*,

*ElementWithUnlimitedResource*,

} *from* './elements/resourcefull';

*import* *Model* *from* './Model';

*import* { *PriorityQueue*, *Queue* } *from* './queues';

*function* *getDefaultModel*(x1*:* number, x2*:* number, x3*:* number) {

*class* Truck {

*constructor*(*private* \_capacity*:* number, *private* \_excavator*:* number) {}

*public* *get* *capacity*() {

*return* *this*.*\_capacity*;

}

*public* *get* *excavator*() {

*return* *this*.*\_excavator*;

}

}

*const getExcavatorDelayGenerator = (*truck*: Truck) => {*

*if (truck.capacity === 20) {*

*return new ExponentialDelayGenerator(5);*

*}*

*if (truck.capacity === 50) {*

*return new ExponentialDelayGenerator(10);*

*}*

*throw new Error(*'Invalid truck capacity'*);*

*}*;

*const getExcavator = (*id*: number) => {*

*const excavator = new ElementWithLimitedResource<Truck>(*

*`EXCAVATOR ${id}`,*

*new Queue(Infinity, [new Truck(50, id)]),*

*{*

*workers: [new Worker(1, getExcavatorDelayGenerator)],*

*},*

*);*

*excavator.inAct(new Truck(20, id));*

*return excavator;*

*}*;

*const excavator1 = getExcavator(1)*;

*const excavator2 = getExcavator(2)*;

*const excavator3 = getExcavator(3)*;

*const going = new ElementWithUnlimitedResource<Truck>(*

'GOING'*,*

*(*id*) =>*

*new Worker(id, (*truck*) => {*

*if (truck.capacity === 20) {*

*return new ConstantDelayGenerator(x1);*

*}*

*if (truck.capacity === 50) {*

*return new ConstantDelayGenerator(x2);*

*}*

*throw new Error(*'Invalid truck capacity'*);*

*}),*

*)*;

*const crushing = new ElementWithLimitedResource<Truck>(*

'CRUSHING'*,*

*new PriorityQueue<Truck>(Infinity, (*a*,* b*) => a.capacity > b.capacity),*

*{*

*workers: [*

*new Worker(1, (*truck*) => {*

*if (truck.capacity === 20) {*

*return new ExponentialDelayGenerator(5);*

*}*

*if (truck.capacity === 50) {*

*return new ExponentialDelayGenerator(4);*

*}*

*throw new Error(*'Invalid truck capacity'*);*

*}),*

*],*

*},*

*)*;

*const goingBack = new ElementWithUnlimitedResource<Truck>(*

'GOING BACK'*,*

*(*id*) =>*

*new Worker(id, (*truck*) => {*

*if (truck.capacity === 20) {*

*return new ConstantDelayGenerator(1.5);*

*}*

*if (truck.capacity === 50) {*

*return new ConstantDelayGenerator(x3);*

*}*

*throw new Error(*'Invalid truck capacity'*);*

*}),*

*)*;

*excavator1*.*next* *=* *new* *SingleNext*({ element: *going* });

*excavator2*.*next* *=* *new* *SingleNext*({ element: *going* });

*excavator3*.*next* *=* *new* *SingleNext*({ element: *going* });

*going*.*next* *=* *new* *SingleNext*({ element: *crushing* });

*crushing*.*next* *=* *new* *SingleNext*({ element: *goingBack* });

*goingBack*.*next* *=* *new* *ConditionalNext*<Truck>()

.*addNextElement*(*excavator1*, (truck) *=>* *truck*.*excavator* *===* 1)

.*addNextElement*(*excavator2*, (truck) *=>* *truck*.*excavator* *===* 2)

.*addNextElement*(*excavator3*, (truck) *=>* *truck*.*excavator* *===* 3);

*const model = new Model([*

*excavator1,*

*excavator2,*

*excavator3,*

*going,*

*crushing,*

*goingBack,*

*])*;

*return* *model*;

}

*const NUMBER\_OF\_RUNS = 20*;

*const SIMULATION\_TIME = 2000*;

*const SKIP\_TIME = 900*;

*for* (*let i = 0*; *i* *<* *NUMBER\_OF\_RUNS*; *i++*) {

*const model = getDefaultModel(0.6, 4.9, 0.1)*;

*model*.*simulate*(*SIMULATION\_TIME*, { skip: *SKIP\_TIME* });

}

Рисунок 4.2 – Приклад коду для проведення експериментів

Таблиця 4.3 – Регресійний аналіз впливу факторів на модель

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **8** | **x0** | **x1** | **x2** | **x3** | **x1x2** | **x1x3** | **x2x3** | **x1x2x3** | **y** |
| **1** | + | + | + | + | + | + | + | + | **1.62** |
| **2** | + | - | + | + | - | - | + | - | **1.89** |
| **3** | + | + | - | + | - | + | - | - | **1.86** |
| **4** | + | - | - | + | + | - | - | + | **2.03** |
| **5** | + | + | + | - | + | - | - | - | **1.78** |
| **6** | + | - | + | - | - | + | - | + | **2.07** |
| **7** | + | + | - | - | - | - | + | + | **2.17** |
| **8** | + | - | - | - | + | + | + | - | **2.44** |

Спираючись на отримані значення, спершу сформуємо рівняння апроксимації відгуку моделі:

(4.2)

Розрахуємо коефіцієнти членів регресії за формулою:

(4.3)

Маємо:

Сформуємо остаточне рівняння регресії:

(4.4)

.

Бачимо, що найбільший вплив на результат чинить саме фактор переїзду до дробівки, зокрема важчого автомобіля, обробка якого займає більше часу загалом. Також, велику долю у фінальному відгуку становить константне значення.

## 5 Інтерпретація результатів моделювання

### 5.1 Інтерпретація отриманих результатів

Дослідимо отримані результати детальніше. З них слідує, що на вихідний відгук системи найбільше впливає час переїзду важчої вантажівки як до дробівки, так і повернення до екскаватору. Бачимо, що вплив групових факторів виражений слабо.

Проте, бачимо, що загалом велику роль у фінальному результаті бере статичний коефіцієнт. Можемо зауважити, що така поведінка є зумовленою замкненістю системи та обмеженістю кількості задач, що можуть надійти до дробівки. Така природа системи зумовлює те, що значних змін у показниках черги не було, проте при цьому бачимо чітку тенденцію на зменшення чи збільшення цього значення залежно від часових умов. Також, на частку сталого значення впливає підбір часових затримок, які необхідні для роботи екскаваторів та самої дробівки, значення, що наведені в умові задачі, досить коректно співвідносяться між собою, через що досліджуваний показник коливається не сильно.

### 5.2 Пропозиції щодо удосконалення системи

Спираючись на інтерпретацію результатів, можемо висунути ряд рекомендацій, що допоможуть системі працювати більш ефективно. Першим та найбільш вагомим вкладом може бути посилення та розширення ресурсів дробівки. Вона приймає на себе навантаження одразу з 3 джерел, що значно підвищує розмір черги перед нею. Не зважаючи на швидкість роботи екскаваторів, вантажівки все ще змушені стояти в очікуванні розвантаження. Таким чином, аби покращити загальну роботу системи можна або доставити додаткові дробівки, або надати більшу кількість кваліфікованого персоналу для прискорення розвантаження. У випадку наявності 2 і більше дробівок рекомендується не розділяти їх за екскаваторами, аби уникнути непотрібних блокувань. Спільні дробівки б могли уникнути простоюванням та значно підсилити систему.

Для підкріплення твердження, проведемо експеримент на уявній покращеній дробівці, що буде опрацьовувати автомобілі швидше. Бачимо, що відгук моделі значно змінився у менший бік.

Таблиця 5.1 – Верифікація зміни фактору при зміні часу обробки дробівки

|  |  |  |  |
| --- | --- | --- | --- |
| **Прогон** | **Вхідні змінні** | | **Вихідні змінні** |
| **t320** | **t350** | **Q4** |
| **1** | exp(5) | exp(4) | 1.7384 |
| **2** | exp(2) | exp(4) | 1.28682 |
| **3** | exp(5) | exp(2) | 1.06552 |

Додатково є сенс вести роботу з екскаваторами та вантажівками теж, адже за умови розширення штату дробівок саме вони можуть стати блокуючим фактором. Якщо екскаватори будуть навантажувати вантажівки, що належать іншому у час, коли їхні автомобілі знаходяться у дробівки, то модель зможе працювати швидше, використовуючи ресурси усіх її елементів.

## Висновок

У процесі виконання курсової роботи було вивчено структуру мережі МО, яка враховує тип обслуговуваних вимог, для визначення можливостей слідування по маршруту та врахування часових затримок. Розроблений алгоритм для моделювання такої системи включав в себе етапи просування модельного часу, просування стану моделі в часі та збір інформації про роботу моделі. Представлена схема реалізації включала опис ключових класів та інтерфейсів.

З використанням спроектованого алгоритму була створена та верифікована програмна модель відповідно до вимог завдання.

Проведені експерименти включали факторний аналіз, що дозволив вивчити вплив часу переїзду та повернення вантажівок різних типів на середню кількість вантажівок у дробівці. Визначили, що найбільший вплив на результат чинить саме фактор переїзду до дробівки, зокрема важчого автомобіля, обробка якого займає більше часу загалом. Також, велику долю у фінальному відгуку становить константне значення. Спираючись на результати, були надані конкретні рекомендації щодо покращення продуктивності системи, зокрема, як приклад, збільшення кількості дробівок або покращення продуктивності наявної.

## Список використаних джерел

* TypeScript [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: <https://www.typescriptlang.org/>.
* Стеценко І. В. Моделювання систем / Інна Вячеславівна Стеценко. – Черкаси: ЧДТУ, 2010. – 407 с.
* Купалова Г. Теорія економічного аналізу / Галина Купалова. – Київ: Знання, 2008. – 639 с.

## Додатки

### Додаток А. Лістинг програмного коду системи

index.ts

*import* { *ExponentialDelayGenerator*, *ConstantDelayGenerator* } *from* './utils';

*import* { *ConditionalNext*, *SingleNext* } *from* './elements/helpers';

*import* {

*Worker*,

*ElementWithLimitedResource*,

*ElementWithUnlimitedResource*,

} *from* './elements/resourcefull';

*import* *Model* *from* './Model';

*import* { *PriorityQueue*, *Queue* } *from* './queues';

*class* Models {

*public* *static* *getDefault*() {

*class* Truck {

*constructor*(*private* \_capacity*:* number, *private* \_excavator*:* number) {}

*public* *get* *capacity*() {

*return* *this*.*\_capacity*;

}

*public* *get* *excavator*() {

*return* *this*.*\_excavator*;

}

}

*const getExcavatorDelayGenerator = (*truck*: Truck) => {*

*if (truck.capacity === 20) {*

*return new ExponentialDelayGenerator(5);*

*}*

*if (truck.capacity === 50) {*

*return new ExponentialDelayGenerator(10);*

*}*

*throw new Error(*'Invalid truck capacity'*);*

*}*;

*const getExcavator = (*id*: number) => {*

*const excavator = new ElementWithLimitedResource<Truck>(*

*`EXCAVATOR ${id}`,*

*new Queue(Infinity, [new Truck(50, id)]),*

*{*

*workers: [new Worker(1, getExcavatorDelayGenerator)],*

*},*

*);*

*excavator.inAct(new Truck(20, id));*

*return excavator;*

*}*;

*const excavator1 = getExcavator(1)*;

*const excavator2 = getExcavator(2)*;

*const excavator3 = getExcavator(3)*;

*const going = new ElementWithUnlimitedResource<Truck>(*

'GOING'*,*

*(*id*) =>*

*new Worker(id, (*truck*) => {*

*if (truck.capacity === 20) {*

*return new ConstantDelayGenerator(2.5);*

*}*

*if (truck.capacity === 50) {*

*return new ConstantDelayGenerator(3);*

*}*

*throw new Error(*'Invalid truck capacity'*);*

*}),*

*)*;

*const crushing = new ElementWithLimitedResource<Truck>(*

'CRUSHING'*,*

*new PriorityQueue<Truck>(Infinity, (*a*,* b*) => a.capacity > b.capacity),*

*{*

*workers: [*

*new Worker(1, (*truck*) => {*

*if (truck.capacity === 20) {*

*return new ExponentialDelayGenerator(5);*

*}*

*if (truck.capacity === 50) {*

*return new ExponentialDelayGenerator(4);*

*}*

*throw new Error(*'Invalid truck capacity'*);*

*}),*

*],*

*},*

*)*;

*const goingBack = new ElementWithUnlimitedResource<Truck>(*

'GOING BACK'*,*

*(*id*) =>*

*new Worker(id, (*truck*) => {*

*if (truck.capacity === 20) {*

*return new ConstantDelayGenerator(1.5);*

*}*

*if (truck.capacity === 50) {*

*return new ConstantDelayGenerator(2);*

*}*

*throw new Error(*'Invalid truck capacity'*);*

*}),*

*)*;

*excavator1*.*next* *=* *new* *SingleNext*({ element: *going* });

*excavator2*.*next* *=* *new* *SingleNext*({ element: *going* });

*excavator3*.*next* *=* *new* *SingleNext*({ element: *going* });

*going*.*next* *=* *new* *SingleNext*({ element: *crushing* });

*crushing*.*next* *=* *new* *SingleNext*({ element: *goingBack* });

*goingBack*.*next* *=* *new* *ConditionalNext*<Truck>()

.*addNextElement*(*excavator1*, (truck) *=>* *truck*.*excavator* *===* 1)

.*addNextElement*(*excavator2*, (truck) *=>* *truck*.*excavator* *===* 2)

.*addNextElement*(*excavator3*, (truck) *=>* *truck*.*excavator* *===* 3);

*const model = new Model([*

*excavator1,*

*excavator2,*

*excavator3,*

*going,*

*crushing,*

*goingBack,*

*])*;

*return* *model*;

}

}

*const model = Models.getDefault()*;

*model*.*simulate*(1000);

Model.ts

*import* *chalk* *from* 'chalk';

*import* { *Element* } *from* './elements';

*import* {

*ElementWithLimitedResource*,

*ElementWithUnlimitedResource*,

} *from* './elements/resourcefull';

*import* *Settings* *from* './Settings';

*type* Row *=* *readonly* (string *|* number)[];

*type* Table *=* Row[];

*interface* SimulationOptions {

*skip:* number;

}

*export* *default* *class* Model<TItem, TElement *extends* Element<TItem>> {

*private* *\_tCurrent:* number;

*private* *\_tNext:* number;

*private* *\_time:* number;

*constructor*(*private* \_list*:* TElement[]) {

*this*.*\_tCurrent* *=* 0;

*this*.*\_tNext* *=* 0;

*this*.*\_time* *=* 0;

}

*public* *simulate*(time*:* number, options*?:* SimulationOptions) {

*const skip = options?.skip ?? 0*;

*this*.*setElementsSkip*(*skip*);

*this*.*\_time* *=* *time*;

*while* (*this*.*\_tCurrent* *<* *time*) {

*this*.*\_tNext* *=* Infinity;

*let currentElement = null*;

*for* (*const element of* *this*.*\_list*) {

*if* (*element*.*tNext* *<* *this*.*\_tNext*) {

*this*.*\_tNext* *=* *element*.*tNext*;

*currentElement* *=* *element*;

}

}

*if* (*currentElement* *===* null) {

*throw* *new* *Error*('There is no current element');

}

*console*.*log*(

`Item in ${*chalk*.*green*(

`'${*currentElement*.*name*}'`,

)}, time = ${*chalk*.*yellow*(*this*.*getFormattedNumber*(*this*.*\_tNext*))}:`,

);

*for* (*const element of* *this*.*\_list*) {

*element*.*calculateStatistics*(*this*.*\_tNext* *-* *this*.*\_tCurrent*);

}

*this*.*\_tCurrent* *=* *this*.*\_tNext*;

*for* (*const element of* *this*.*\_list*) {

*element*.*tCurrent* *=* *this*.*\_tCurrent*;

}

*currentElement*.*outAct*();

*for* (*const element of* *this*.*\_list*) {

*if* (*element*.*tNext* *===* *this*.*\_tCurrent*) {

*element*.*outAct*();

}

}

*this*.*printInformation*();

}

*this*.*printResults*();

}

*private* *setElementsSkip*(skip*:* number) {

*this*.*\_list*.*forEach*((element) *=>* {

*element*.*skip* *=* *skip*;

});

}

*private* *printInformation*() {

*const table: (string | number)[][] = [[*'Name'*,* 'Quantity'*,* 'Time next'*]]*;

*for* (*const element of* *this*.*\_list*) {

*table*.*push*([

*element*.*name*,

*...this*.*getFormattedRow*(*element*.*getInformation*()),

]);

}

*console*.*table*(*table*);

*console*.*log*();

}

*private* *printResults*() {

*console*.*log*('Results');

*const table: Table = [*

*[*

'Name'*,*

'Quantity'*,*

'Mean work time'*,*

'Mean time before in'*,*

'Mean time before out'*,*

'Mean queue size'*,*

'Failures number'*,*

'Failure probability'*,*

*],*

*]*;

*for* (*const element of* *this*.*\_list*) {

*const commonColumns = [element.name, element.quantity] as const*;

*if* (*element* *instanceof* ElementWithLimitedResource) {

*table*.*push*([

*...commonColumns*,

*...this*.*getFormattedRow*(

*this*.*printResultsElementWithLimitedResources*(*element*),

),

]);

} *else* *if* (*element* *instanceof* ElementWithUnlimitedResource) {

*table*.*push*([

*...commonColumns*,

*...this*.*getFormattedRow*(

*this*.*printResultsForProcessWithUnlimitedResources*(*element*),

),

]);

}

}

*console*.*table*(*table*);

}

*private* *printResultsElementWithLimitedResources*(

element*:* ElementWithLimitedResource<TItem>,

) {

*const meanWorkingTime = element.workingTime / this.\_time*;

*const meanTimeBeforeIn = element.totalTimeBeforeIn / this.\_time*;

*const meanTimeBeforeOut = element.totalTimeBeforeOut / this.\_time*;

*const failuresProbability =*

*element.failuresNumber / (element.quantity + element.failuresNumber)*;

*const meanQueueSize = element.queuesSizes / this.\_time*;

*return* [

*meanWorkingTime*,

*meanTimeBeforeIn*,

*meanTimeBeforeOut*,

*meanQueueSize*,

*element*.*failuresNumber*,

*failuresProbability*,

] *as* *const*;

}

*private* *printResultsForProcessWithUnlimitedResources*(

element*:* ElementWithUnlimitedResource<TItem>,

) {

*const meanWorkingTime = element.workingTime / this.\_time*;

*const meanTimeBeforeIn = element.totalTimeBeforeIn / this.\_time*;

*const meanTimeBeforeOut = element.totalTimeBeforeOut / this.\_time*;

*return* [

*meanWorkingTime*,

*meanTimeBeforeIn*,

*meanTimeBeforeOut*,

'-',

'-',

'-',

] *as* *const*;

}

*private* *getFormattedRow*(row*:* Row) {

*return* *row*.*map*((value) *=>*

*typeof* *value* *===* 'number' *?* *this*.*getFormattedNumber*(*value*) *:* *value*,

);

}

*private* *getFormattedNumber*(number*:* number) {

*if* (*number* *%* 1 *!==* 0) {

*return* *Number*(*number*.*toFixed*(*Settings*.*FLOAT\_PRECISION*));

}

*return* *number*;

}

}

Settings.ts

*export* *default* *class* Settings {

*public* *static* *readonly* *FLOAT\_PRECISION* *=* 5;

}

Random.ts

*export* *default* *class* Random {

*public* *static* *getExponential*(mean*:* number) {

*let number = 0*;

*while* (*number* *===* 0) {

*number* *=* *Math*.*random*();

}

*number* *=* *-mean* *\** *Math*.*log*(*number*);

*return* *number*;

}

*public* *static* *getUniform*(min*:* number, max*:* number) {

*let number = 0*;

*while* (*number* *===* 0) {

*number* *=* *Math*.*random*();

}

*number* *=* *min* *+* *number* *\** (*max* *-* *min*);

*return* *number*;

}

*public* *static* *getGaussian*() {

*let number = 0*;

*for* (*let i = 0*; *i* *<* 6; *i* *+=* 1) {

*number* *+=* *Math*.*random*();

}

*return* *number* */* 6;

}

*public* *static* *getNormal*(mean*:* number, deviation*:* number) {

*return* *mean* *+* *Random*.*getGaussian*() *\** *deviation*;

}

*public* *static* *getErlang*(mean*:* number, deviation*:* number) {

*let number = 0*;

*for* (*let i = 0*; *i* *<* *deviation*; *i++*) {

*number* *+=* *Random*.*getExponential*(*mean*);

}

*return* *-*1 */* *deviation* */* *mean* *+* *Math*.*log*(*number*);

}

*public* *static* *getBoolean*() {

*return* *Math*.*random*() *>* 0.5;

}

}

DelayGenerator.ts

*import* *Random* *from* './Random';

*export* *interface* DelayGenerator {

*get*()*:* number;

}

*export* *class* ConstantDelayGenerator *implements* DelayGenerator {

*constructor*(*private* \_delay*:* number) {}

*public* *get*() {

*return* *this*.*\_delay*;

}

}

*export* *class* UniformDelayGenerator *implements* DelayGenerator {

*constructor*(*private* \_min*:* number, *private* \_max*:* number) {}

*public* *get*() {

*return* *Random*.*getUniform*(*this*.*\_min*, *this*.*\_max*);

}

}

*export* *class* ExponentialDelayGenerator *implements* DelayGenerator {

*constructor*(*private* \_mean*:* number) {}

*public* *get*() {

*return* *Random*.*getExponential*(*this*.*\_mean*);

}

}

*export* *class* NormalDelayGenerator *implements* DelayGenerator {

*constructor*(*private* \_mean*:* number, *private* \_deviation*:* number) {}

*public* *get*() {

*return* *Random*.*getNormal*(*this*.*\_mean*, *this*.*\_deviation*);

}

}

*export* *class* ErlangDelayGenerator *implements* DelayGenerator {

*constructor*(*private* \_mean*:* number, *private* \_deviation*:* number) {}

*public* *get*() {

*return* *Random*.*getErlang*(*this*.*\_mean*, *this*.*\_deviation*);

}

}

*export* *class* GaussianDelayGenerator *implements* DelayGenerator {

*public* *get*() {

*return* *Random*.*getGaussian*();

}

}

Queue.ts

*export* *default* *class* Queue<TItem> {

*constructor*(

*protected* \_capacity*:* number *=* 0,

*protected* \_items*:* TItem[] *=* [],

) {}

*public* *get* *capacity*() {

*return* *this*.*\_capacity*;

}

*public* *get* *size*() {

*return* *this*.*\_items*.length;

}

*public* *peek*() {

*return* *this*.*\_items*[0];

}

*public* *push*(item*:* TItem) {

*if* (*this*.*isFull*) {

*throw* *new* *Error*('Queue is full');

}

*this*.*\_items*.*push*(*item*);

*return* *item*;

}

*public* *pop*() {

*if* (*this*.*isEmpty*) {

*throw* *new* *Error*('Queue is empty');

}

*return* *this*.*\_items*.*shift*();

}

*public* *get* *isFull*() {

*return* *this*.*\_items*.length *===* *this*.*\_capacity*;

}

*public* *get* *isEmpty*() {

*return* *this*.*\_items*.length *===* 0;

}

}

PriorityQueue.ts

*import* *Queue* *from* './Queue';

*export* *default* *class* PriorityQueue<TItem> *extends* Queue<TItem> {

*protected* *static* *readonly* *TOP\_INDEX* *=* 0;

*constructor*(

\_capacity*:* number *=* 0,

*private* *\_comparator* *=* (a*:* TItem, b*:* TItem) *=>* *a* *>* *b*,

\_items*:* TItem[] *=* [],

) {

*super*(*\_capacity*, *\_items*);

}

*public* *push*(item*:* TItem) {

*this*.*\_items*.*push*(*item*);

*this*.*shiftUp*();

*return* *item*;

}

*public* *pop*() {

*const poppedValue = this.peek()*;

*const bottom = this.size - 1*;

*if* (*bottom* *>* *PriorityQueue*.*TOP\_INDEX*) {

*this*.*swap*(*PriorityQueue*.*TOP\_INDEX*, *bottom*);

}

*this*.*\_items*.*pop*();

*this*.*shiftDown*();

*return* *poppedValue*;

}

*protected* *isGreater*(i*:* number, j*:* number) {

*return* *this*.*\_comparator*(*this*.*\_items*[*i*], *this*.*\_items*[*j*]);

}

*protected* *swap*(i*:* number, j*:* number) {

[*this*.*\_items*[*i*], *this*.*\_items*[*j*]] *=* [*this*.*\_items*[*j*], *this*.*\_items*[*i*]];

}

*protected* *shiftUp*() {

*let node = this.size - 1*;

*while* (

*node* *>* *PriorityQueue*.*TOP\_INDEX* *&&*

*this*.*isGreater*(*node*, *this*.*getParentIndex*(*node*))

) {

*this*.*swap*(*node*, *this*.*getParentIndex*(*node*));

*node* *=* *this*.*getParentIndex*(*node*);

}

}

*protected* *shiftDown*() {

*let node = PriorityQueue.TOP\_INDEX*;

*while* (

(*this*.*getLeftIndex*(*node*) *<* *this*.*size* *&&*

*this*.*isGreater*(*this*.*getLeftIndex*(*node*), *node*)) *||*

(*this*.*getRightIndex*(*node*) *<* *this*.*size* *&&*

*this*.*isGreater*(*this*.*getRightIndex*(*node*), *node*))

) {

*let maxChild =*

*this.getRightIndex(node) < this.size &&*

*this.isGreater(this.getRightIndex(node), this.getLeftIndex(node))*

*?* *this*.*getRightIndex*(*node*)

*:* *this*.*getLeftIndex*(*node*);

*this*.*swap*(*node*, *maxChild*);

*node* *=* *maxChild*;

}

}

*protected* *getParentIndex*(index*:* number) {

*return* ((*index* *+* 1) *>>>* 1) *-* 1;

}

*protected* *getLeftIndex*(index*:* number) {

*return* (*index* *<<* 1) *+* 1;

}

*protected* *getRightIndex*(index*:* number) {

*return* (*index* *+* 1) *<<* 1;

}

}

Element.ts

*import* { *Next* } *from* './helpers';

*export* *default* *abstract* *class* Element<TItem> {

*private* *static* *NEXT\_ID* *=* 0;

*private* *\_id:* number;

*private* *\_tCurrent:* number;

*private* *\_tNext:* number;

*private* *\_quantity:* number;

*private* *\_totalTimeBeforeIn:* number;

*private* *\_tInPrevious:* number;

*private* *\_totalTimeBeforeOut:* number;

*private* *\_tOutPrevious:* number;

*private* *\_skip:* number;

*private* *\_next:* Next<TItem> *|* null;

*constructor*(*private* \_name*:* string) {

*this*.*\_id* *=* *Element*.*NEXT\_ID++*;

*this*.*\_tCurrent* *=* 0;

*this*.*\_tNext* *=* Infinity;

*this*.*\_quantity* *=* 0;

*this*.*\_totalTimeBeforeIn* *=* 0;

*this*.*\_tInPrevious* *=* 0;

*this*.*\_totalTimeBeforeOut* *=* 0;

*this*.*\_tOutPrevious* *=* 0;

*this*.*\_next* *=* null;

*this*.*\_skip* *=* 0;

}

*public* *abstract* *get* *isFree*()*:* boolean;

*public* *calculateStatistics*(delta*:* number) {}

*public* *inAct*(item*:* TItem *|* null) {

*if* (*!this*.*shouldSkip*()) {

*this*.*\_totalTimeBeforeIn* *+=* *this*.*tCurrent* *-* *this*.*\_tInPrevious*;

*this*.*\_tInPrevious* *=* *this*.*tCurrent*;

}

}

*public* *outAct*() {

*if* (*!this*.*shouldSkip*()) {

*this*.*\_totalTimeBeforeOut* *+=* *this*.*tCurrent* *-* *this*.*\_tOutPrevious*;

*this*.*\_tOutPrevious* *=* *this*.*tCurrent*;

*this*.*\_quantity++*;

}

}

*public* *get* *quantity*() {

*return* *this*.*\_quantity*;

}

*public* *get* *totalTimeBeforeIn*() {

*return* *this*.*\_totalTimeBeforeIn*;

}

*public* *get* *totalTimeBeforeOut*() {

*return* *this*.*\_totalTimeBeforeOut*;

}

*public* *get* *tCurrent*() {

*return* *this*.*\_tCurrent*;

}

*public* *set* *tCurrent*(tCurrent*:* number) {

*this*.*\_tCurrent* *=* *tCurrent*;

}

*public* *get* *tNext*() {

*return* *this*.*\_tNext*;

}

*protected* *set* *tNext*(tNext*:* number) {

*this*.*\_tNext* *=* *tNext*;

}

*public* *get* *id*() {

*return* *this*.*\_id*;

}

*protected* *set* *id*(id*:* number) {

*this*.*\_id* *=* *id*;

}

*public* *get* *name*() {

*return* *this*.*\_name*;

}

*protected* *set* *name*(name*:* string) {

*this*.*\_name* *=* *name*;

}

*public* *get* *next*() {

*return* *this*.*\_next*;

}

*public* *set* *next*(next*:* Next<TItem> *|* null) {

*this*.*\_next* *=* *next*;

}

*public* *set* *skip*(skip*:* number) {

*this*.*\_skip* *=* *skip*;

}

*public* *get* *skip*() {

*return* *this*.*\_skip*;

}

*public* *shouldSkip*() {

*return* *this*.*\_skip* *>=* *this*.*tCurrent*;

}

*public* *getInformation*() {

*return* [*this*.*quantity*, *this*.*tNext*] *as* *const*;

}

}

Worker.ts

*import* { *DelayGenerator* } *from* '../../utils';

*export* *enum* WorkerState {

*FREE* *=* 'free',

*BUSY* *=* 'busy',

}

*export* *default* *class* Worker<TItem> {

*private* *\_state:* WorkerState;

*private* *\_item:* TItem *|* null;

*private* *\_tNext:* number;

*constructor*(

*private* \_id*:* number,

*private* *\_getDelayGenerator:* (item*:* TItem) *=>* DelayGenerator,

) {

*this*.*\_state* *=* *WorkerState*.*FREE*;

*this*.*\_tNext* *=* Infinity;

*this*.*\_item* *=* null;

}

*public* *get* *id*() {

*return* *this*.*\_id*;

}

*public* *get* *tNext*() {

*return* *this*.*\_tNext*;

}

*public* *set* *tNext*(tNext*:* number) {

*this*.*\_tNext* *=* *tNext*;

}

*public* *get* *state*() {

*return* *this*.*\_state*;

}

*public* *set* *state*(state*:* WorkerState) {

*this*.*\_state* *=* *state*;

}

*public* *get* *item*() {

*return* *this*.*\_item*;

}

*public* *set* *item*(item*:* TItem *|* null) {

*this*.*\_item* *=* *item*;

}

*public* *getDelay*() {

*if* (*!this*.*\_item*) {

*throw* *new* *Error*('No item');

}

*return* *this*.*\_getDelayGenerator*(*this*.*\_item*).*get*();

}

}

Next.ts

*import* *Element* *from* '../Element';

*export* *interface* NextElement<TItem> {

*element:* Element<TItem>;

*withBlocking?:* boolean;

}

*export* *interface* Next<TItem> {

*getNextElement*(item*:* TItem)*:* NextElement<TItem> *|* null;

}

*export* *class* SingleNext<TItem> *implements* Next<TItem> {

*constructor*(*private* \_nextElement*:* NextElement<TItem> *|* null) {}

*public* *getNextElement*() {

*return* *this*.*\_nextElement*;

}

}

*export* *class* RandomNext<TItem> *implements* Next<TItem> {

*constructor*(*private* \_nextElements*:* NextElement<TItem>[]) {}

*public* *getNextElement*() {

*return* *this*.*\_nextElements*.*at*(*this*.*getRandomIndex*()) *??* null;

}

*private* *getRandomIndex*() {

*return* *Math*.*floor*(*Math*.*random*() *\** *this*.*\_nextElements*.length);

}

}

*export* *interface* NextElementWithPriority<TItem> *extends* NextElement<TItem> {

*priority:* number;

}

*export* *class* PrioritizedNext<TItem> *implements* Next<TItem> {

*constructor*(*private* \_nextElements*:* NextElementWithPriority<TItem>[]) {}

*public* *getNextElement*() {

*return* *this*.*getNextElementsSortedByPriority*().*at*(0) *??* null;

}

*private* *getNextElementsSortedByPriority*() {

*return* *this*.*\_nextElements*.*toSorted*((a, b) *=>* *b*.*priority* *-* *a*.*priority*);

}

}

*export* *interface* NextElementWithProbability<TItem> *extends* NextElement<TItem> {

*probability:* number;

}

*export* *class* ProbabilisticNext<TItem> *implements* Next<TItem> {

*constructor*(*private* \_nextElements*:* NextElementWithProbability<TItem>[]) {}

*public* *getNextElement*() {

*const random = Math.random()*;

*if* (*this*.*getNextElementsProbabilitiesSum*() *!==* 1) {

*throw* *new* *Error*('Sum of probabilities is not equal to 1');

}

*let probabilitiesSum = 0*;

*for* (*const nextElement of* *this*.*\_nextElements*) {

*probabilitiesSum* *+=* *nextElement*.*probability*;

*if* (*random* *<=* *probabilitiesSum*) {

*return* *nextElement*;

}

}

*return* null;

}

*protected* *getNextElementsProbabilitiesSum*() {

*return* *this*.*\_nextElements*.*reduce*(

(sum, { probability }) *=>* *sum* *+* *probability*,

0,

);

}

}

*interface* NextElementWithCondition<TItem> *extends* NextElement<TItem> {

*condition:* (item*:* TItem) *=>* boolean;

}

*export* *class* ConditionalNext<TItem> *implements* Next<TItem> {

*constructor*(*private* \_nextElements*:* NextElementWithCondition<TItem>[] *=* []) {}

*public* *getNextElement*(item*:* TItem) {

*return* *this*.*\_nextElements*.*find*(({ condition }) *=>* *condition*(*item*)) *??* null;

}

*public* *addNextElement*(

element*:* Element<TItem>,

*condition:* (item*:* TItem) *=>* boolean,

) {

*this*.*\_nextElements*.*push*({ *element*, *condition* });

*return* *this*;

}

}

ElementWithResource.ts

*import* *Element* *from* '../Element';

*import* *Worker*, { *WorkerState* } *from* './Worker';

*export* *default* *abstract* *class* ElementWithResource<

TItem,

> *extends* Element<TItem> {

*private* *\_workers:* Worker<TItem>[];

*private* *\_workingTime:* number;

*constructor*(name*:* string) {

*super*(*name*);

*this*.*\_workers* *=* [];

*this*.*\_workingTime* *=* 0;

}

*protected* *get* *workers*() {

*return* *this*.*\_workers*;

}

*protected* *set* *workers*(workers*:* Worker<TItem>[]) {

*this*.*\_workers* *=* *workers*;

}

*public* *get* *workingTime*() {

*return* *this*.*\_workingTime*;

}

*protected* *set* *workingTime*(workingTime*:* number) {

*this*.*\_workingTime* *=* *workingTime*;

}

*protected* *getFreeWorkers*() {

*return* *this*.*workers*.*filter*((worker) *=>* *worker*.*state* *===* *WorkerState*.*FREE*);

}

*protected* *getFreeWorker*() {

*return* (

*this*.*workers*.*find*((worker) *=>* *worker*.*state* *===* *WorkerState*.*FREE*) *??* null

);

}

*protected* *getBusyWorkers*() {

*return* *this*.*workers*.*filter*((worker) *=>* *worker*.*state* *===* *WorkerState*.*BUSY*);

}

*protected* *getBusyWorkerWithMinimumTNext*() {

*const busyWorkers = this.getBusyWorkers()*;

*if* (*!busyWorkers*.length) {

*return* null;

}

*let minimumTNext = Infinity*;

*let busyWorkerWithMinimumTNext = null*;

*for* (*const busyWorker of* *busyWorkers*) {

*const { tNext } = busyWorker*;

*if* (*tNext* *<* *minimumTNext*) {

*busyWorkerWithMinimumTNext* *=* *busyWorker*;

*minimumTNext* *=* *tNext*;

}

}

*return* *busyWorkerWithMinimumTNext*;

}

*protected* *getMinimumTNextFromBusyWorkers*() {

*const busyWorker = this.getBusyWorkerWithMinimumTNext()*;

*if* (*!busyWorker*) {

*return* Infinity;

}

*return* *busyWorker*.*tNext*;

}

*protected* *removeWorker*(worker*:* Worker<TItem>) {

*this*.*workers* *=* *this*.*workers*.*filter*(

(currentWorker) *=>* *currentWorker*.*id* *!==* *worker*.*id*,

);

}

}

ElementWithUnlimitedResource.ts

*import* *Worker*, { *WorkerState* } *from* './Worker';

*import* *ElementWithResource* *from* './ElementWithResource';

*export* *default* *class* ElementWithUnlimitedResource<

TItem,

> *extends* ElementWithResource<TItem> {

*private* *static* *NEXT\_WORKER\_ID* *=* 0;

*constructor*(

name*:* string,

*private* *\_createWorker:* (id*:* number) *=>* Worker<TItem>,

) {

*super*(*name*);

}

*public* *inAct*(item*:* TItem) {

*super*.*inAct*(*item*);

*const freeWorker =*

*this.getFreeWorker() ??*

*this*.*\_createWorker*(*ElementWithUnlimitedResource*.*NEXT\_WORKER\_ID++*);

*if* (*!this*.*workers*.*includes*(*freeWorker*)) {

*this*.*workers*.*push*(*freeWorker*);

}

*freeWorker*.*state* *=* *WorkerState*.*BUSY*;

*freeWorker*.*item* *=* *item*;

*const delay = freeWorker.getDelay()*;

*freeWorker*.*tNext* *=* *this*.*tCurrent* *+* *delay*;

*if* (*!this*.*shouldSkip*()) {

*this*.*workingTime* *+=* *delay*;

}

*this*.*tNext* *=* *this*.*getMinimumTNextFromBusyWorkers*();

}

*public* *outAct*() {

*super*.*outAct*();

*const busyWorker = this.getBusyWorkerWithMinimumTNext()*;

*if* (*!busyWorker*) {

*throw* *new* *Error*('There is no busy workers');

}

*const item = busyWorker.item*;

*if* (*!item*) {

*throw* *new* *Error*('There is no item in busy worker');

}

*const nextElement = this.next?.getNextElement(item)*;

*if* (

*nextElement* *&&*

*!nextElement*.*element*.*isFree* *&&*

*nextElement*.*withBlocking*

) {

*busyWorker*.*tNext* *=* *nextElement*.*element*.*tNext*;

*return*;

}

*this*.*removeWorker*(*busyWorker*);

*this*.*tNext* *=* *this*.*getMinimumTNextFromBusyWorkers*();

*nextElement*?.*element*.*inAct*(*item*);

}

*public* *get* *isFree*() {

*return* true;

}

}

ElementWithLimitedResource.ts

*import* { *Queue* } *from* '../../queues';

*import* *Worker*, { *WorkerState* } *from* './Worker';

*import* *ElementWithResource* *from* './ElementWithResource';

*export* *interface* ElementWithLimitedResourceOptions<TItem> {

*workers:* Worker<TItem>[];

}

*export* *default* *class* ElementWithLimitedResource<

TItem,

> *extends* ElementWithResource<TItem> {

*private* *\_failuresNumber:* number;

*private* *\_queuesSizes:* number;

*constructor*(

name*:* string,

*private* \_queue*:* Queue<TItem>,

{ workers }*:* ElementWithLimitedResourceOptions<TItem>,

) {

*super*(*name*);

*this*.*workers* *=* *workers*;

*this*.*\_queuesSizes* *=* 0;

*this*.*\_failuresNumber* *=* 0;

}

*public* *inAct*(item*:* TItem) {

*super*.*inAct*(*item*);

*const freeWorker = this.getFreeWorker()*;

*if* (*freeWorker*) {

*freeWorker*.*state* *=* *WorkerState*.*BUSY*;

*freeWorker*.*item* *=* *item*;

*const delay = freeWorker.getDelay()*;

*freeWorker*.*tNext* *=* *this*.*tCurrent* *+* *delay*;

*if* (*!this*.*shouldSkip*()) {

*this*.*workingTime* *+=* *delay*;

}

*this*.*tNext* *=* *this*.*getMinimumTNextFromBusyWorkers*();

*return*;

}

*if* (*!this*.*\_queue*.*isFull*) {

*this*.*\_queue*.*push*(*item*);

*return*;

}

*if* (*!this*.*shouldSkip*()) {

*this*.*\_failuresNumber++*;

}

}

*public* *outAct*() {

*super*.*outAct*();

*const busyWorker = this.getBusyWorkerWithMinimumTNext()*;

*if* (*!busyWorker*) {

*throw* *new* *Error*('There is no busy workers');

}

*const item = busyWorker.item*;

*if* (*!item*) {

*throw* *new* *Error*('There is no item in busy worker');

}

*busyWorker*.*tNext* *=* Infinity;

*busyWorker*.*state* *=* *WorkerState*.*FREE*;

*busyWorker*.*item* *=* null;

*this*.*tNext* *=* *this*.*getMinimumTNextFromBusyWorkers*();

*const nextElement = this.next?.getNextElement(item)*;

*if* (

*nextElement* *&&*

*!nextElement*.*element*.*isFree* *&&*

*nextElement*.*withBlocking*

) {

*busyWorker*.*tNext* *=* *nextElement*.*element*.*tNext*;

*busyWorker*.*item* *=* *item*;

*busyWorker*.*state* *=* *WorkerState*.*BUSY*;

*return*;

}

*if* (*!this*.*\_queue*.*isEmpty*) {

*const item = this.\_queue.pop()!*;

*busyWorker*.*state* *=* *WorkerState*.*BUSY*;

*busyWorker*.*item* *=* *item*;

*const delay = busyWorker.getDelay()*;

*busyWorker*.*tNext* *=* *this*.*tCurrent* *+* *delay*;

*if* (*!this*.*shouldSkip*()) {

*this*.*workingTime* *+=* *delay*;

}

*this*.*tNext* *=* *this*.*getMinimumTNextFromBusyWorkers*();

}

*nextElement*?.*element*.*inAct*(*item*);

}

*public* *calculateStatistics*(delta*:* number) {

*if* (*!this*.*shouldSkip*()) {

*this*.*\_queuesSizes* *+=* *this*.*\_queue*.*size* *\** *delta*;

}

}

*public* *get* *isFree*() {

*return* *Boolean*(*this*.*getFreeWorker*());

}

*public* *get* *queue*() {

*return* *this*.*\_queue*;

}

*public* *set* *queue*(queue*:* Queue<TItem>) {

*this*.*\_queue* *=* *queue*;

}

*public* *get* *failuresNumber*() {

*return* *this*.*\_failuresNumber*;

}

*public* *get* *queuesSizes*() {

*return* *this*.*\_queuesSizes*;

}

}