

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: «Імітаційна модель функціонування приймального відділення
лікарні на основі формального опису мережею масового
обслуговування»

Керівник:

Асистент Дифучин Антон Юрійович

«Допущено до захисту»

«___» _____ 2024 р.

Захищено з оцінкою

Члени комісії:

Виконавець:

Мешков Андрій Ігорович

студент групи ПІ-15

залікова книжка № ПІ-1522

«05» грудня 2024 р.

Інна СТЕЦЕНКО

Антон ДИФУЧИН

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет Інформатики та обчислювальної техніки

Кафедра Інформатики та програмної інженерії

Дисципліна Технології паралельних обчислень

Курс 4 Група ІІІ 15 Семестр 4

ЗАВДАННЯ

на курсову роботу студента

Мешкова Андрія Ігоровича

1. Тема роботи Імітаційна модель функціонування приймального відділення лікарні на основі формального опису мережею масового обслуговування

2. Термін здачі студентом закінченої роботи "05" грудня 2024 р.

3. Вихідні дані до проекту

Варіант №А14

У лікарню надходять хворі таких трьох типів: 1) хворі, що пройшли попереднє обстеження і направлені на лікування; 2) хворі, що бажають потрапити в лікарню, але не пройшли цілком попереднє обстеження; 3) хворі, які тільки що поступили на попереднє обстеження. Час обслуговування хворих різноманітних типів у приймальному відділенні розподілені відповідно до таблиці 2.5.

Таблиця 2.5. Числові значення до завдання А14.

<i>Тип хворого</i>	<i>Відносна частота</i>	<i>Середній час реєстрації, хв</i>
1	0,9 до 10.00 години та 0,5 після 10.00 години	15
2	0,1 у будь-який час від 0.00 до 10.00 години	40
3	0,4 після 10.00 години	20

При надходженні у приймальне відділення хворий стає в чергу, якщо обидва чергових лікарі зайняті. Лікар, що звільнився, вибирає в першу чергу тих хворих, що вже пройшли попереднє обстеження. Після заповнення різноманітних форм у приймальне відділення хворі 1 типу ідуть прямо в палату, а хворі типів 2 і 3 направляються в лабораторію. Троє супровідних розводять хворих по палатах. Хворим не дозволяється направлятися в палату без супровідного. Якщо всі супровідні зайняті, хворі очікують їхнього звільнення в приймальному відділенні. Як тільки хворий доставлений у палату, він вважається тим, що завершив процес прийому до лікарні. Супровідному потребується 3 хвилини для того, щоб повернутися в приймальне відділення після доставки хворого в палату. Хворі, що спрямовуються в лабораторію, не потребують супроводу. Після прибуття в лабораторію хворі стають у чергу в реєстратуру. Після реєстрації вони ідуть у кімнату очікування, де чекають виклику до одного з двох лаборантів. Після здачі аналізів хворі або повертаються в приймальне відділення (якщо їх приймають у лікарню), або залишають лікарню (якщо їм було призначено тільки попереднє обстеження). Після повернення в приймальне відділення хворий, що здав аналізи, розглядається як хворий типу 1. Приймальне відділення відкрите з 7.00 до 17.00. Проте попередні обстеження (тип 3) не призначаються до 10.00 через ранкове переобтяження лабораторії. Хворі, що надходять після 16.00, направляються в профілакторій. Проте хворі типу 2, що повертаються з лабораторії, приймаються до 17.00, тобто доти, поки чергові лікарі не закінчать роботу і приймальне відділення не закриється. У таблиці 2.6 приводяться дані по тривалості дій (у хвилинах).

Таблиця 2.6. Розподіли випадкових величин до завдання A14.

<i>Величина</i>	<i>Розподіл</i>
Час між прибуттями в приймальне відділення	Експоненціальний з математичним сподіванням 15 хвилин
Час проходження в палату	Рівномірний від 3 до 8 хвилин
Час проходження з приймального відділення в лабораторію або з лабораторії в приймальне відділення	Рівномірний від 2 до 5 хвилин

Час обслуговування в реєстратуру лабораторії	Ерланга з математичним сподіванням 4,5 хвилини і $k=3$
Час проведення аналізу в лабораторії	Ерланга з математичним сподіванням 4 хвилини і $k=2$

Визначити час, проведений хворим у системі, тобто інтервал часу, починаючи з надходження і закінчуючи доставкою в палату (для хворих типу 1 і 2) або виходом із лабораторії (для хворих типу 3). Визначити також інтервал між прибуттями хворих у лабораторію. Припустити, що всі черги використовують упорядкування за стратегією FIFO і мають необмежену довжину.

4. Для вищезазначеної системи:

- 1) розробити опис концептуальної моделі системи;
- 2) виконати формалізацію опису об'єкта моделювання в термінах визначеного у завданні формалізму;
- 3) розробити алгоритм імітації моделі дискретно-подійної системи у відповідності до побудованого формального опису;
- 4) для доведення коректності побудованого алгоритму виконати верифікацію алгоритму імітації;
- 5) визначити статистичні оцінки заданих характеристик моделі, що є метою моделювання;
- 6) провести експериментальне дослідження моделі;
- 7) інтерпретувати результати моделювання та сформулювати пропозиції щодо поліпшення функціонування системи;
- 8) зробити висновки щодо складності розробки моделі та алгоритму імітації на основі використаного формалізму, отриманих результатів моделювання та їх корисності.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень)

Графічного матеріалу не має.

6. Дата видачі завдання "05" листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів курсового проекту (роботи)	Термін виконання етапів роботи	Примітка
1	Отримання завдання	05.11.2024	
2	Формулювання теми курсової роботи	06.11.2024	
3	Аналіз можливих методів вирішення поставленого завдання	08.11.2024	
4	Розробка концептуальної моделі	10.11.2024	
5	Опис імітаційної моделі	12.11.2024	
6	Опис програмної реалізації імітаційної моделі	14.11.2024	
7	Аналіз та оцінка результатів	22.11.2024	
8	Оформлення пояснювальної записки	24.11.2024	
9	Здача пояснювальної записки	27.11.2024	
10	Захист курсового проекту (роботи)	05.12.2024	

Студент _____

(підпис)

Керівник _____

(підпис)

Дифучин А.Ю.

(прізвище, ім'я, по батькові)

«05» листопада 2024 р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 60 сторінок, 16 рисунків, 6 таблиць, 10 формул.

Об'єкт дослідження: універсальний алгоритм імітації, в якому можна враховувати різний тип об'єктів обслуговування. Часові затримки та переходи в маршрутах слідування між елементами можуть встановлюватись в залежності від типу об'єкта.

Мета роботи: розробити універсальний алгоритм імітації, верифікувати алгоритм на власному прикладі, провести експерименти на мережі МО.

Ключові слова: імітаційна модель, алгоритм імітації.

ЗМІСТ

ПОСТАНОВКА ЗАВДАННЯ	8
ВСТУП.....	9
1. КОНЦЕПТУАЛЬНА МОДЕЛЬ.....	11
1.1 АНАЛІЗ МОЖЛИВИХ МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ПОСТАВЛЕНОГО ЗАВДАННЯ	11
1.2 ЦІЛІ МОДЕЛЮВАННЯ	12
1.3 СХЕМАТИЧНЕ ПРЕДСТАВЛЕННЯ ПРОЦЕСУ МОДЕЛЮВАННЯ.....	12
1.4 ВХІДНІ ТА ВИХІДНІ ЗМІННІ	14
2. ФОРМАЛІЗОВАНА МОДЕЛЬ	15
2.1 ФОРМАЛІЗМ МЕРЕЖ МАСОВОГО ОБСЛУГОВУВАННЯ.....	15
2.2 ФОРМАЛІЗОВАНЕ ПРЕДСТАВЛЕННЯ ПРОЦЕСУ МОДЕЛЮВАННЯ	18
2.3 ОБЧИСЛЕННЯ ВИХІДНИХ ЗМІННИХ.....	20
2.3.1 <i>Параметри, що розраховуються під час симуляції</i>	20
2.3.2 <i>Параметри, що обчислюються теоретично</i>	21
3. РЕАЛІЗАЦІЯ МОДЕЛІ.....	23
3.1 ВИБІР ЗАСОБУ АВТОМАТИЗАЦІЇ.....	23
3.2 РЕАЛІЗАЦІЯ АЛГОРИТМУ ІМІТАЦІЇ.....	23
3.3 РЕАЛІЗАЦІЯ ЕЛЕМЕНТІВ МЕРЕЖІ МАСОВОГО ОБСЛУГОВУВАННЯ.....	26
3.3.1 <i>Реалізація каналів</i>	26
3.3.2 <i>Реалізація створювача та процесів</i>	26
3.3.3 <i>Реалізація переходу</i>	30
3.4 ОБЧИСЛЕННЯ ВИХІДНИХ ХАРАКТЕРИСТИК КОДУ.....	31
3.5 ПОБУДОВА МОДЕЛІ	33
3.6 ВЕРИФІКАЦІЯ МОДЕЛІ	35
4. ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ НА МОДЕЛІ	36
4.1 ТАКТИЧНЕ ПЛАНУВАННЯ	36
4.2 СТРАТЕГІЧНЕ ПЛАНУВАННЯ.....	38
5. ІНТЕРПРЕТАЦІЯ РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ.....	41
ВИСНОВКИ.....	43
ДЖЕРЕЛА	45
ДОДАТОК А	46

ПОСТАНОВКА ЗАВДАННЯ

Основною задачею курсового проекту є розробка моделі, що імітує роботу приймального відділення лікарні у формалізмі мереж масового обслуговування.

Система має імітувати обслуговування пацієнтів трьох типів у приймальному відділенні лікарні, що надходять для обстеження, лікування або очікують направлення до лабораторії. Зокрема:

1. Хворі першого типу пройшли попереднє обстеження і потребують швидкого оформлення для направлення в палати. У разі відсутності супроводжуючого персоналу, пацієнти очікують в приймальному відділенні, поки супроводжуючий не звільниться.

2. Хворі другого типу не завершили попереднє обстеження і направляються до лабораторії. Після завершення обстеження, вони можуть бути прийняті до лікарні, якщо приймальне відділення ще працює.

3. Хворі третього типу лише прибувають для попереднього обстеження, проходять реєстрацію, і після аналізів або направляються до палати, або залишають лікарню.

Система має моделювати всі етапи обслуговування в приймальному відділенні, включаючи час очікування, реєстрацію, розподіл до палат або лабораторії, обробку супроводжуючими, а також обслуговування та повернення з лабораторії.

ВСТУП

Тема курсової роботи — «Імітаційна модель функціонування приймального відділення лікарні на основі формального опису мережею масового обслуговування» — присвячена моделюванню складної системи обслуговування, що дозволяє вивчити особливості функціонування приймального в умовах змінного навантаження та обмежених ресурсів. Завданням дослідження є розробка імітаційної моделі, що відтворює процес обслуговування пацієнтів різних типів у приймальному відділенні лікарні з подальшим направленням до палат чи лабораторії.

Актуальність дослідження обумовлена потребою оптимізувати роботу приймальних відділень медичних закладів, що важливо для підвищення ефективності обслуговування пацієнтів і зменшення часу очікування в чергах. Такий підхід дає можливість точно прогнозувати навантаження на приймальне відділення в різний час доби, оптимально розподілити наявні ресурси.

Мета дослідження — визначити ключові характеристики системи, такі як середній час перебування пацієнта в системі, інтервал між надходженням хворих у лабораторію. На основі отриманих результатів планується розробити рекомендації щодо покращення роботи приймального відділення.

Для вирішення поставлених задач у курсовій роботі використовується формалізм мереж масового обслуговування. Це дозволяє точно описати процеси обслуговування та руху пацієнтів у межах приймального відділення, визначити залежності між потоками хворих різних типів та можливості їх обслуговування лікарями і супроводжуючим персоналом.

Методи дослідження включають:

1. Дискретно-подійну імітацію, яка дозволяє змоделювати події, що виникають у приймальному відділенні, з урахуванням випадкових факторів, таких як час прибуття пацієнтів і тривалість обслуговування.
2. Математичний аналіз результатів імітації для обчислення середніх

значень та ймовірнісних характеристик, що описують роботу приймального відділення.

Для реалізації моделі та виконання експериментів з її налаштування використовується мова програмування Java, що забезпечує зручні засоби для дослідження поведінки складних систем.

У результаті дослідження будуть зроблені висновки щодо ефективності роботи приймального відділення, а також сформульовані пропозиції для оптимізації організації обслуговування пацієнтів у приймальних відділеннях на основі отриманих даних.

1. КОНЦЕПТУАЛЬНА МОДЕЛЬ

1.1 Аналіз можливих методів та засобів вирішення поставленого завдання

Модель — це абстрактне представлення об'єкта або системи, що полегшує науковий аналіз та дослідження [1, с. 10]. Основними методами моделювання є аналітичне, математичне та імітаційне. Аналітичне моделювання полягає в поданні системи у вигляді аналітичної залежності змінних Y від X . Якщо немає можливості отримати розв'язок аналітичним шляхом, але існує алгоритм для точного розв'язання задачі, застосовують математичний метод. У випадку, коли система не піддається аналітичному опису, використовують імітацію, тобто відтворення роботи системи за допомогою комп'ютерної програми [1, с. 12-13].

З огляду на завдання, імітаційне моделювання може бути найбільш прийнятним рішенням з таких причин:

- Складність системи: Управління процесами лікування та обробка пацієнтів може передбачати значну кількість змінних та взаємодіючих параметрів, які важко формалізувати аналітично. Імітаційне моделювання дозволяє врахувати складні аспекти виробничого процесу, не вдаючись до спрощень чи припущень.

- Невизначеність у параметрах: Деякі параметри, як-от часові розподіли, визначаються з певною похибкою або невизначеністю, що складно реалізувати аналітичними методами.

- Змінність параметрів: В реальних умовах можуть відбуватися зміни у кількості персоналу та швидкості їх роботи, кількість хворих, які надходять. Імітаційне моделювання дозволяє враховувати динамічні зміни системи, реагуючи на них у реальному часі.

- Відстеження динаміки подій: Імітаційне моделювання дає змогу прослідковувати послідовність подій у часі, враховуючи взаємодію різних частин системи. Це важливо для вивчення динамічної природи процесів та їх впливу на загальну ефективність роботи системи.

- Можливість експериментів зі сценаріями: Імітаційне моделювання надає змогу тестувати різні сценарії та стратегії управління, що допомагає визначити оптимальні рішення та оцінити їхній вплив на систему.

Отже, зважаючи на складність і динамічність виробничих процесів, які потрібно розглянути в цьому завданні, імітаційне моделювання забезпечує більш гнучкий і адаптивний підхід для дослідження та оптимізації приймального відділення.

1.2 Цілі моделювання

Метою моделювання є створення імітаційної моделі роботи приймального відділення лікарні у формалізмі системи масового обслуговування (СМО). Модель дозволить аналізувати основні показники ефективності системи, зокрема загальний час перебування пацієнта у приймальному відділенні та інтервали між прибуттям пацієнтів.

1.3 Схематичне представлення процесу моделювання

Для ефективного аналізу роботи приймального відділення лікарні у формалізмі мережі масового обслуговування було створено схему, яка відображає ключові етапи та взаємодії між ними. Схема на рисунку 1.1 демонструє послідовність обробки пацієнтів різних типів, від моменту прибуття до завершення процесу прийому.

У системі приймального відділення пацієнти надходять трьома основними потоками:

1. Пацієнти, що пройшли попереднє обстеження (тип 1) – ці пацієнти проходять реєстрацію у приймальному відділенні та після оформлення направляються безпосередньо в палати.

2. Пацієнти без попереднього обстеження (тип 2) – після реєстрації ці пацієнти спрямовуються до лабораторії для проведення необхідних аналізів.

3. Пацієнти, що прибули на попереднє обстеження (тип 3) – цей тип

пацієнтів починає надходити до приймального відділення після 10:00. Після оформлення вони також спрямовуються до лабораторії.

На рисунку 1.1 зображено, як пацієнти стають в чергу у приймальному відділенні. Після завершення реєстрації пацієнти типу 1 вирушають до палати, супроводжувані персоналом. Пацієнти типів 2 та 3 прямують до лабораторії, де вони спершу проходять реєстрацію, а потім очікують виклику до одного з лаборантів для здачі аналізів.

Після завершення аналізів пацієнти:

- Типу 2 повертаються до приймального відділення, де вони вважаються готовими до прийому в лікарню, якщо приймальня ще працює.

- Типу 3 залишають систему після виходу з лабораторії, якщо вони проходили лише попереднє обстеження.

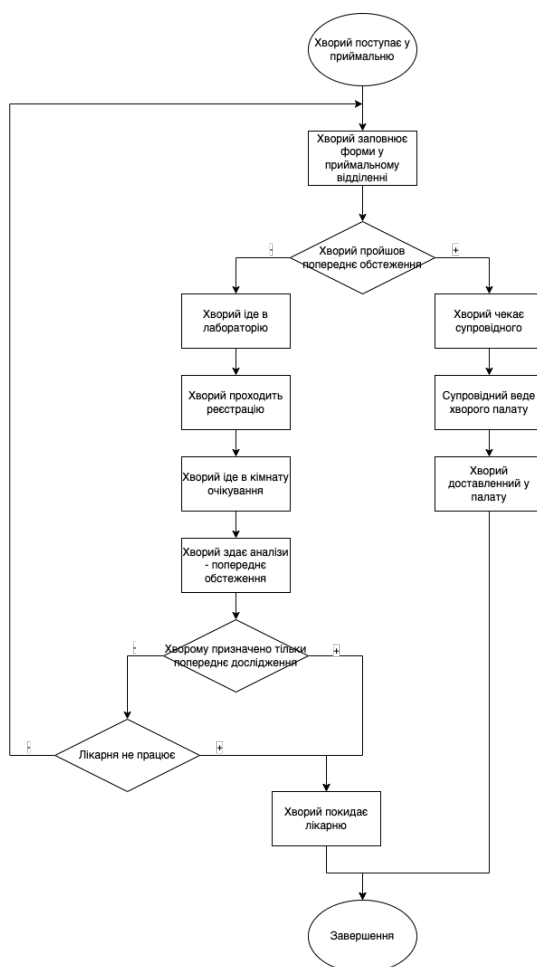


Рисунок 1.1 - Схематичне представлення процесу роботи моделі

Рисунок 1.1 ілюструє всі етапи, через які проходить пацієнт від моменту

прибуття до завершення обслуговування у приймальному відділенні, забезпечуючи структуроване уявлення про процес прийому в лікарню для різних типів пацієнтів.

1.4 Вхідні та вихідні змінні

Вхідними змінними моделі є такі параметри, що визначають роботу приймального відділення лікарні та потоки пацієнтів. Зокрема, це:

- Час прибуття пацієнтів до приймального відділення.
- Тип пацієнта (визначає маршрут проходження і пріоритет обслуговування в черзі):

1. Пацієнти, які пройшли попереднє обстеження.
2. Пацієнти, що потребують первинного обстеження.
3. Пацієнти, що надійшли на попереднє обстеження.

- Час обслуговування пацієнтів у приймальному відділенні.
- Час переведення пацієнтів до палат.
- Час переведення пацієнтів до лабораторії та з лабораторії до приймального відділення.

- Час обслуговування в реєстратурі лабораторії.
- Час проведення аналізу в лабораторії.

Вихідними змінними моделі, які необхідно визначити, є:

- Загальний час перебування пацієнта у системі – включає період від прибуття до приймального відділення до моменту його направлення в палату (для пацієнтів типу 1 і 2) або виходу з лабораторії (для пацієнтів типу 3).

- Інтервал між прибуттями пацієнтів до лабораторії – для аналізу завантаження лабораторії та ефективності обслуговування.

Ці вихідні змінні дозволяють оцінити загальний час очікування та проходження обслуговування у системі, що є важливим для визначення ефективності роботи приймального відділення лікарні.

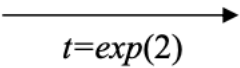
2. ФОРМАЛІЗОВАНА МОДЕЛЬ

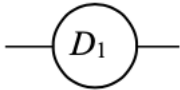
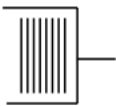
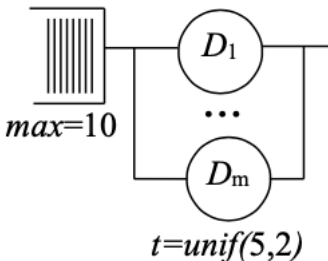

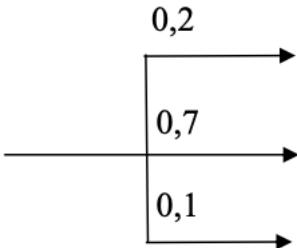
2.1 Формалізм мереж масового обслуговування

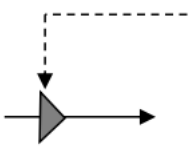
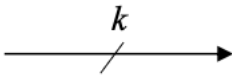
Мережа масового обслуговування (МО) – це комплекс систем масового обслуговування (СМО), між якими об'єкти переміщуються з певною ймовірністю. Кожна СМО складається з одного або декількох каналів (пристроїв), що обробляють об'єкт за визначений період часу. Заявки, що надходять для обслуговування, проходять через послідовність СМО. При надходженні в СМО заявка потрапляє на вільний канал, а за його відсутності – у чергу. Якщо черга переповнена, реєструється відмова. Важливо, що протягом процесу обслуговування заявка не зникає, не об'єднується з іншими заявками і не розщеплюється на частини [1, с. 63-65].

Для опису МО використовуються схематичне представлення, елементи якого описані в таблиці 2.1.

Таблиця 2.1. Елементи формального опису мережі масового обслуговування [1, с.52-54]

Елемент	Графічне позначення	Зміст	Числові параметри
Надходження запитів ззовні		Надходження запитів на обслуговування з заданою часовою затримкою. Якщо надходження запитів ззовні немає, то цей елемент може бути відсутнім.	t - часова затримка в умовних одиницях часу, між послідовними надходженнями запитів на обслуговування, яка може бути вказана детермінованим значенням або випадковою величиною з заданим законом розподілу. Для мережі без зовнішнього надходження має бути вказана кількість запитів, що циркулюють в ній.

Елемент	Графічне позначення	Зміст	Числові параметри
Пристрій обслуговування	 $t = \text{unif}(5, 2)$	Пристрій, названий D_1 , обробляє запити по одному з часовою затримкою t	t - часова затримка в умовних одиницях часу, необхідна для обробки одного запиту, яка може бути вказана детермінованим значенням або випадковою величиною з заданим законом розподілу
Черга	 $\text{max}=10$	Накопичувач запитів, що очікують звільнення пристроя обслуговування.	max – обмеження на кількість місць в черзі, яке може бути задане цілим числом. За замовчуванням (тобто якщо параметр не вказаний), кількість місць в черзі не обмежена.
Система масового обслуговування (СМО)	 $t = \text{unif}(5, 2)$	Один або кілька ідентичних пристроїв з чергою перед ними. Черга є складовою частиною СМО, тому з'єднання між чергою та пристроями не дуга, а відрізок.	m - кількість пристроїв у СМО
Дуга		Маршрут слідування запиту до наступної СМО	За замовчуванням ймовірність слідування по маршруту 1.
Розгалуження маршруту		Маршрут слідування запиту до однієї з наступних СМО, вибір якої здійснюється за заданими ймовірностями.	Ймовірності вибору кожного маршруту слідування, які у сумі складають 1.

Елемент	Графічне позначення	Зміст	Числові параметри
Блокування маршруту		<p>Встановлює блокування маршруту слідування у наступну СМО, якщо $B(S,t)=1$. Запит залишається у пристрої, з якого намагається вийти. Пристрій переходить у стан блокований (час обробки запиту завершився, але залишити пристрій немає можливості). Маршрут відкритий тільки за умови $B(S,t)=0$.</p>	<p>$B(S, t)$ – предикат, який в найбільш загальному випадку залежить від стану S моделі та/або поточному моменту часу t. Наприклад, якщо умовою блокування маршруту є досягнення черги свого обмеження L_{\max}, то предикат має вигляд $B(S, t) = (L_j(t) = L_{\max})$.</p>
Дуга з числовим параметром		<p>Групуювання елементів у вказаній кількості</p>	<p>Якщо на дузі вказаний параметр k, то k запитів перетворюються на дузі в 1 запит (групуються), що спрямовується далі по маршруту. Якщо на дузі вказаний параметр $1/k$, то 1 запит перетворюється у k запитів (розгруповується), що спрямовуються далі по маршруту.</p>

Отже: розробимо МО, що імітує роботу приймального відділення лікарні.

2.2 Формалізоване представлення процесу моделювання

Об'єктом обслуговування є хворі трьох типів:

1. Хворі, що пройшли попереднє обстеження і направлені на лікування. Вони одразу реєструються і після цього прямують у палату.

2. Хворі, які бажають потрапити до лікарні, але не пройшли повне попереднє обстеження. Після реєстрації вони прямують до лабораторії. Проходять всі процеси до тестів, після цього з тестами прямують до приймальної, і проходять процеси як хворі типу 1.

3. Хворі, які тільки що поступили на попереднє обстеження. Вони реєструються і направляються в лабораторію.

Модель передбачає такі етапи обслуговування:

1. Реєстрація в приймальному відділенні:

- Реєстрацію здійснюють два чергових лікарі.
- Якщо обидва лікарі зайняті, хворі очікують у черзі (FIFO).
- Пріоритет обслуговування: хворі 1-го типу обслуговуються першими.

2. Направлення до палати:

- Після реєстрації хворі 1-го типу чекають звільнення одного з трьох супровідних.
- Якщо всі лікарі зайняті, хворі очікують у черзі (FIFO).

3. Направлення до лабораторії:

- Хворі типів 2 і 3 після реєстрації прямують у лабораторію без супроводу.
- В даній СМО черги немає, тому всі прямують до наступної.

4. Реєстрація в лабораторії:

- Хворі очікують у черзі (FIFO).
- Проходять реєстрацію і переходять далі.

5. Здача аналізів в лабораторії:

- Кличуть з черги два лаборанти.
- Хворі сидять у кімнаті очікування (FIFO).

6. Після здачі аналізів хворі:

- Типу 2 – повертаються в приймальне відділення з результатами тестів(isPassed) як хворі типу 1, звідки прямують у палату(якщо приймальне відділення ще працює).
- Типу 3 – залишають лікарню.

Графік роботи:

- Приймальне відділення: з 7:00 до 17:00.
- Попереднє обстеження (хворі типу 3): з 10:00 до 16:00.
- Після 16:00 нових хворих направляють до профілакторію, окрім тих, хто повернувся з лабораторії (типу 2).

Розподіли часу:

- Прибуття в приймальне відділення, $t_{\text{надх}}$: випадкова величина, розподілена за експоненціальним законом з математичним сподіванням 15 хвилин.
- Реєстрація хворих:
 - Тип 1, $t_{\text{обр1}}$: детерміноване число 15 хвилин.
 - Тип 2, $t_{\text{обр2}}$: детерміноване число 40 хвилин.
 - Тип 3, $t_{\text{обр3}}$: детерміноване число 20 хвилин.
- Перехід до палати, $t_{\text{обр}}$: випадкова величина, рівномірно розподілена в інтервалі (3, 8), або (5.5-2.5, 5.5+2.5) хвилин.
- Повернення супровідного, або час очікування, $t_{\text{оч}}$: детерміноване число 3 хвилини.
- Перехід між відділеннями, $t_{\text{обр}}$: випадкова величина, рівномірно розподілена в інтервалі (2, 5).
- Реєстрація в лабораторії: випадкова величина, розподілена за законом Ерланга з математичним сподіванням 4.5 та $k=3$.
- Аналізи в лабораторії: випадкова величина, розподілена за законом Ерланга з математичним сподіванням 4 та $k=2$.

Початкові умови:

- Усі черги необмежені (FIFO).

- У приймальному відділенні є два лікарі.
- Для супроводу до палат доступні три супровідних.

Завдання:

1. Час у системі:

- Обчислити час перебування хворого в системі. Типи 1 і 2: від прибуття до доставки в палату. Тип 3: від прибуття до виходу з лабораторії.

2. Інтервал прибуття в лабораторію:

- Визначити час між прибуттями хворих у лабораторію.

Таким чином можемо створити схему системи у формалізмі МО (рис.2.1)

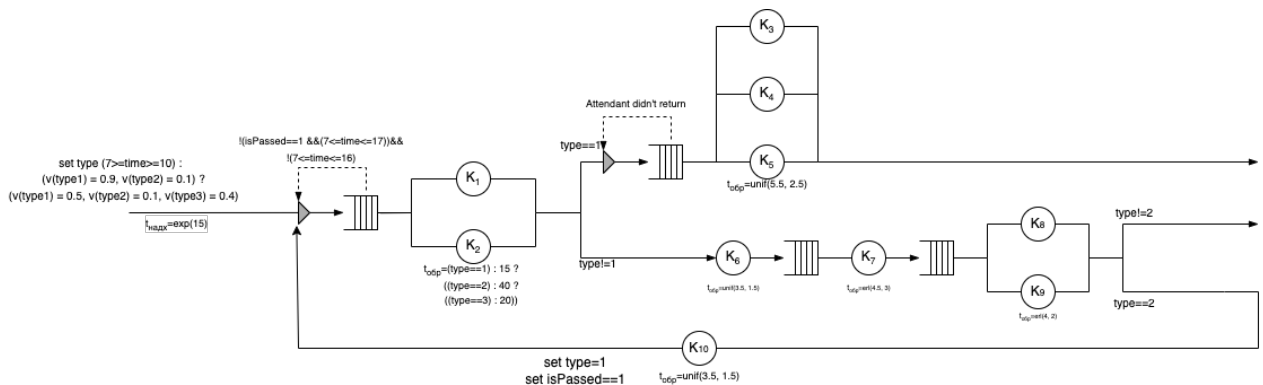


Рисунок 2.1 - Схема формалізованого представлення системи

2.3 Обчислення вихідних змінних

Для виконання завдання визначимо, які параметри розраховуються під час симуляції, а які можна обчислити за допомогою формул.

2.3.1 Параметри, що розраховуються під час симуляції

1. Час, проведений хворими в системі:

- Для хворих типу 1 і 2 – від моменту надходження до завершення доставки в палату.
- Для хворих типу 3 – від моменту надходження до завершення обслуговування в лабораторії.
- Цей час залежить від часу очікування в чергах, часу обслуговування

(реєстрації, здачі аналізів, доставки в палату), тривалості переходів між відділеннями.

2. Інтервали між прибуттями хворих у лабораторію:

- Інтервали визначаються випадковим розподілом, з урахуванням черг у приймальному відділенні та графіка роботи.

Для кожного значення вираховується середнє значення та квадратичне відхилення за формулами 2.1 і 2.2.

$$y_i = \frac{\sum_{s=1}^n y_{is}}{n} \quad (2.1)$$

де y_s – спостережуване значення величини.

$$\sigma^2 = \frac{\sum_{s=1}^n (y_{is} - \bar{y})^2}{n} \quad (2.2)$$

де, y_s – спостережуване значення величини, \bar{y} – середнє значення.

2.3.2 Параметри, що обчислюються теоретично

Час між прибуттями хворих у приймальне відділення визначається з експоненційного розподілу із середнім значенням 15 хвилин (формула 2.1).

$$t = -\lambda \ln(\zeta) \quad (2.3)$$

де λ – середній час прибуття, ζ – випадкове число (0,1).

Час переходу між відділеннями визначається з рівномірного розподілу (формула 2.2).

$$t = a + (b - a) \cdot \zeta \quad (2.4)$$

де a , b — мінімальне і максимальне значення тривалості переходу, ζ — випадкове число (0,1).

Час обслуговування в лабораторії – розподіл Ерланга (формула 2.3).

$$t = -\frac{1}{\lambda} \sum_{i=1}^k \ln(\zeta_i) \quad (2.5)$$

де k — кількість фаз (2 для аналізу, 3 для реєстрації), λ – середнє значення для кожної фази, ζ — випадкове число (0,1).

3. РЕАЛІЗАЦІЯ МОДЕЛІ

3.1 Вибір засобу автоматизації

В якості засобу автоматизації було вирішено створити власну розфарбовану мережу МО. Це є краще ніж використання готового рішення, оскільки дозволяє вносити потрібні зміни в модель та в разі необхідності розраховувати потрібні характеристики. В такому випадку є максимально простим організація збирання статистики.

В якості мови програмування було обрано Java. Java надає потужні засоби для реалізації об'єктно-орієнтованих моделей, що дозволяє більш ефективно відображати структуру реальної моделі МО. Java дозволяє легко генерувати випадкові числа і керувати ними, що важливо для імітаційного моделювання. [3]

Для генерації чисел за рівномірним, експоненційним розподілом чи розподілом Ерланга було реалізовано власні методи, які використовують метод `Math.random()`.

Для вирішення даної задачі графічний інтерфейс не є необхідним, а отже можна використовувати консольний вивід.

3.2 Реалізація алгоритму імітації

Алгоритм імітації – це цикл, що виконує ітерації системи, доки час не перевищить час моделювання. Ітерація системи містить такі кроки: знайти найближчу подію, просунути час, виконати найближчі події. Отже, блок-схема алгоритму виглядає так (рис. 3.1).



Рисунок 3.1 - Блок-схема алгоритму імітації

Для реалізації алгоритму створимо клас Model, який ініціюється елементами, що входять в систему. Реалізуємо метод симуляції, який приймає загальний час симуляції (рис. 3.2).

```

public void simulate(double time) {
    while (this.tCurr < time) {
        this.tNext = Double.MAX_VALUE;
        for (Element e : list) {
            if ((e.getTnext() < this.tNext)) {
                this.tNext = e.getTnext();
            }
        }
        for (Element e : list) {
            e.doStatistics(tNext - tCurr);
        }
        this.tCurr = this.tNext;
        for (Element e : list) {
            e.setTcurr(tCurr);
        }
        for (Element e : list) {
            if (e.getTnext() == this.tCurr) {
                e.outAct();
            }
        }
    }
    printResult();
}
  
```

Рисунок 3.2 - Реалізація алгоритму імітації

У таблиці 3.1 наведено опис головних атрибутів та методів, що використовуються.

Таблиця 3.1 - Опис атрибутів та методів класу Model

Атрибут / метод	Опис	Початкове значення
tcurr	Поточний час в симуляції	0
tnext	Час наступної події	0
list	Елементи моделі	[]
simulate	Імітація моделі	
doStatistic()	Розрахунок потрібної статистики	
printResult()	Виведення результатів	

Функція виводу результату зображена на рисунку 3.3

```
public void printResult() {
    System.out.println(x:"\n-----RESULTS-----");
    for (Element e : list) {
        e.printResult();
    }
}
```

Рисунок 3.3 - Функція виводу результату

3.3 Реалізація елементів мережі масового обслуговування

3.3.1 Реалізація каналів

Для обробки вимог створимо клас Device (канал). Конструктор класу приймає значення середнього часу обробки та метод розподілу. Клас містить методи inAct (реалізує додавання вимоги в обробку – об’єкт переходить в статус працюючий), outAct (реалізує вивільнення вимоги з обробки – об’єкт переходить в статус вільний).

3.3.2 Реалізація створювача та процесів

Модель розроблена у пункті 3.2 використовує об’єкти типу Element. Element – це абстрактний клас, що описує спільні атрибути і методи для створювача та процесу.

Element ініціюється іменем, масивами середніх значень та їх розподілів, з яких створюється масив каналів.

Головні атрибути і методи класу Element наведені у таблиці 3.2.

Таблиця 3.2 - Опис атрибутів та методів класу Element

Атрибут / метод	Опис	Початкове значення
name	Назва елементу	
id	ID елементу	
devices	Масив каналів	
transfer	Опціональний клас переходу до наступної СМО	
tcurr	Поточний час симуляції	0
tnext	Обчислювана змінна, що повертає мінімальний час наступної події з каналів	
quantity	Кількість хворих пройшовших процес.	
inAct()	Метод передачі вимоги до елементу	
outAct()	Метод виходу вимоги	
doStatistic()	Метод розрахунку статистики	
printResult()	Виведення результатів	

Клас Create наслідує клас Element. Клас Create містить лише один канал. Перевантажений метод outAct() після виходу вимоги одразу додає нову вимогу у канал (рис. 3.5). Створення класу Patient було необхідною дією для полегшення контролю присвоювання та зміни типу хворого, також клас зберігає у собі час входу в систему та час виходу. Клас Transfer для переходу між процесами буде описаний пізніше.

```

@Override
public void outAct( ) {
    super.outAct();
    Time time = new Time(super.getTcurr());
    double hours = time.getHours();
    double p1,p2,p3;
    if(hours >= 7 && hours <= 10 ){
        p1 = 0.9;
        p2 = 0.1;
        p3 = 0;
    } else {
        p1 = 0.5;
        p2 = 0.1;
        p3 = 0.4;
    }

    Patient newPatient = generatePatient(p1, p2, p3);
    newPatient.setArrival(super.getTcurr());
    double delay = 0;
    for (Device device : super.getDevices()) {
        device.setPatient(newPatient);
        delay = device.getDelayAverage(index:0);
    }
    Transfer transfer = super.getTransfer();
    String next = transfer.goNext(this, super.getTcurr(), newPatient);

    if((next == "blocked")){
        super.setTnext(super.getTcurr() + time.getNightDelay());
    } else {
        super.setTnext(super.getTcurr() + delay);
    }
}

```

Рисунок 3.4 - Перевантаження методу outAct() у класі Create

Клас Process наслідує клас Element. Клас Process містить чергу та перевантажує обчислювану змінну queue. Клас Process перевантажує метод inAct(). Якщо елемент вільний, то вимога передається в перший вільний канал.

Клас Process перевантажує метод outAct(). З кожного пристрою, який завершив обробку виймаються вимоги (викликається outAct() для пристрою). Для всіх вимог робиться спроба зробити перехід до наступного елементу. При наявності вимог у черзі, вони додаються у вільний канал (рис. 3.5).

```

@Override
public void outAct( ) {
    List<Patient> outTasks = new ArrayList<>();
    for (Device device : super.getDevices()) {
        device.setTcurr(super.getTcurr());
        if (device.getTNext() == super.getTcurr()) {
            Patient task = device.outAct();
            if(task instanceof Patient && task != null) {
                outTasks.add(task);
            }
        }
    }
    if (outTasks.size() > 0){
        for (Patient task : outTasks) {
            super.outAct();
            Transfer transfer = super.getTransfer();
            if(transfer != null){
                String next = transfer.goNext(this, super.getTcurr(), task);
                if((next != "blocked") && queue.length() > 0){
                    Patient patient = this.queue.remove();
                    int type = patient.getType();
                    for (Device device : super.getDevices()) {
                        device.setTcurr(super.getTcurr());
                        if (device.getState() == 0) {
                            int delayNumber = device.getDelayNumber();
                            if(delayNumber == 3){
                                device.inAct(patient, type-1);
                            } else {
                                device.inAct(patient, index:0);
                            }
                        }
                        super.updateTnext();
                        return;
                    }
                }
                if(next == null){
                    this.addDuration(task);
                }
            }
            if(transfer == null){
                this.addDuration(task);
            }
        }
    }
    for (Device device : super.getDevices()) {
        if (device.getState() == 0) {
            device.setTNext(Double.MAX_VALUE);
        }
    }
    super.updateTnext();
}

```

Рисунок 3.5 - Перевантаження методу outAct() у класі Process

Як було визначено у пункті 2.3.1 нам треба розраховувати час, проведений хворими в системі та інтервал між прибуттям хворих у лабораторію. Отже, якщо СМО в даний момент має статус працююча, то додаємо `delta`(змінна різниці часу, що передається у метод), якщо у цей час хворий закінчив роботу системі, у масив додається час проведення в системі, та `delta` додається в масив інтервалів даного хворого. Система має 3 види хворих, які проходять різну кількість процесів, тому існує 3 масиви часу в системі, і по масиву інтервалів для кожного. Також кожний раз коли хворий заходить у лабораторію, записується значення в масив, так само `delta`. Потім масиви `delta` будуть потрібні для розрахунку середнього значення і квадратичного відхилення.

3.3.3 Реалізація переходу

Для реалізації переходу створено інтерфейс `Transfer`, що описує метод `goNext()`. Функція `goNext()` шукає наступний елемент за допомогою `getIndex()`, де перевіряються умови переходу чи блокування та зміни(зміна типу і флагу хворого 2 при поверненні до приймальної), якщо перехід не заблокований, викликає `inAct()` у наступного елемента та повертає статус, вільно, заблоковано або `null` якщо далі нема процесів – хворий виходить з системи (рис. 3.6). Детальніше код наведений у додатку А.

```

import java.util.List;

public class Transfer {
    private List<Process> nextElements;

    public Transfer(List<Process> nextElements){
        this.nextElements = nextElements;
    }

    public String goNext(Process current, double tCurr, Patient patient){
        int index = this.getIndex(current, tCurr, patient);
        if(index >= 0){
            Process nextElement = nextElements.get(index);
            nextElement.inAct(patient);
            return "free";
        } else if (index == -1){
            return "blocked";
        } else {
            return null;
        }
    }

    public String goNext(Create current, double tCurr, Patient patient){
        Time time = new Time(tCurr);
        double hours = time.getHours();

        if(hours >= 7 && hours <= 16){
            Process nextElement = nextElements.get(index:0);
            nextElement.inAct(patient);
            return "free";
        } else {
            return "blocked";
        }
    }
}

```

Рисунок 3.6 - Клас Transfer

3.4 Обчислення вихідних характеристик коду

Для обчислення часу хворого в системі хворий запам'ятовує час входу в систему, на виході з системи(коли Transfer дорівнює нулю, бо нема наступних процесів, або goNext() повертає нуль, бо є можливі процеси далі, але даний хворий не відповідає умовам) фіксується час, вираховується різниця виходу і входу, дане число додається в масив, залежно від свого типу.

Для обчислення інтервалів між зверненням на вході inAct() у реєстрацію лабораторії, час входу кожного хворого додається в масив. В кінці імітації масив вхідних точок перераховується у масив інтервалів.

У методі `printResult` вираховується середнє та квадратичне відхилення усіх вихідних даних (рис. 3.7-3.8).

```
public void printResult(){
    System.out.println("\n" + this.name+ "\nquantity = "+ quantity);
    if(type1.size() > 0){
        double average = average(type1);
        double variance = variance(average, type1);
        double stdDev = Math.sqrt(variance);
        System.out.println("\nType 1: Average Duration = " + average + ", Standard Deviation = " + stdDev);
    }
    if(type2.size() > 0){
        double average = average(type2);
        double variance = variance(average, type2);
        double stdDev = Math.sqrt(variance);
        System.out.println("\nType 2: Average Duration = " + average + ", Standard Deviation = " + stdDev);
    }
    if(type3.size() > 0){
        double average = average(type3);
        double variance = variance(average, type3);
        double stdDev = Math.sqrt(variance);
        System.out.println("\nType 3: Average Duration = " + average + ", Standard Deviation = " + stdDev);
    }

    if(entryTimes.size()>0){
        List<Double> intervals = new ArrayList<>();
        for (int i = 1; i < entryTimes.size(); i++) {
            double interval = entryTimes.get(i) - entryTimes.get(i - 1);
            intervals.add(interval);
        }

        double average = average(intervals);
        double variance = variance(average, intervals);
        double stdDev = Math.sqrt(variance);
        System.out.println("\nAverage Interval = " + average + ", Standard Deviation = " + stdDev);
    }
}
```

Рисунок 3.7 – Метод `printResult`

```
public double average(List<Double> list){
    return list.stream().mapToDouble(Double::doubleValue).average().orElse(0);
}

public double variance(double average, List<Double> list){
    return list.stream()
        .mapToDouble(x -> Math.pow(x - average, 2))
        .average()
        .orElse(0);
}
```

Рисунок 3.8 – Обчислення середнього числа та квадратичного відхилення

3.5 Побудова моделі

Для побудови моделі треба створити клас в якому визначити потрібні параметри для виводу. Потрібно створити створювач з одним каналом та заданим інтервалом. Створити процеси для приймальні, переходу в палату, переходу в лабораторію, реєстрації в лабораторії, здачі тестів та повернення в приймальне відділення. Для кожного з процесів потрібно вказати правильний розподіл та задати додаткові параметри. Додати можливі переходи, умови входу у даний процес, або умови блокування, умови зміни(зміна типу хворого).

Код побудови моделі наведено на рисунку 3.9. Результат виконання коду наведено на рисунку 3.10.

```
public class App {
    Run | Debug
    public static void main(String[] args){

        Create arrivalOfPatients = new Create(nameOfElement:"Arrival of patients", Arrays.asList(...a:15.0), Arrays.asList(...a:"exp"), numberOfDevices:1);

        Process reception = new Process(nameOfElement:"Reception", Arrays.asList(...a:15.0, 40.0, 20.0), Arrays.asList(...a:"", "", ""), numberOfDevices:2);

        Process transitionWard = new Process(nameOfElement:"Transition to the ward", Arrays.asList(...a:3.0, 3.0), Arrays.asList(...a:"unif", ""), numberOfDevices:3);
        transitionWard.setDelay(index:0, delayDev:8.0);

        Process referralLaboratory = new Process(nameOfElement:"Referral to the laboratory", Arrays.asList(...a:2.0), Arrays.asList(...a:"unif", ""), numberOfDevices:1);
        referralLaboratory.setDelay(index:0, delayDev:5.0);

        Process registrationLaboratory = new Process(nameOfElement:"Registration in the laboratory", Arrays.asList(...a:4.5), Arrays.asList(...a:"erl", ""), numberOfDevices:1);
        registrationLaboratory.setDelay(index:0, delayDev:3.0);

        Process passingLaboratory = new Process(nameOfElement:"Passing tests in the laboratory", Arrays.asList(...a:19.0), Arrays.asList(...a:"erl", ""), numberOfDevices:2);
        passingLaboratory.setDelay(index:0, delayDev:2.0);

        Process returningRecaption= new Process(nameOfElement:"Returning to reception", Arrays.asList(...a:2.0), Arrays.asList(...a:"unif", ""), numberOfDevices:1);
        passingLaboratory.setDelay(index:0, delayDev:5.0);

        arrivalOfPatients.setTransfer(Arrays.asList(reception));

        reception.setBlockers(new String[][]{{"flag", "1", "time", "7", "16"}, {"time", "7", "17"}});
        reception.setTransfer(Arrays.asList(transitionWard, referralLaboratory));

        transitionWard.setConditions(new String[][]{{"type", "1"}});

        referralLaboratory.setConditions(new String[][]{{"!type", "1"}});
        referralLaboratory.setTransfer(Arrays.asList(registrationLaboratory));

        registrationLaboratory.setTransfer(Arrays.asList(passingLaboratory));

        passingLaboratory.setTransfer(Arrays.asList(returningRecaption));

        returningRecaption.setConditions(new String[][]{{"type", "2"}});
        returningRecaption.setTransfer(Arrays.asList(reception));
        returningRecaption.setChanges(new String[][]{{"type", "1"}, {"flag", "1"}});

        ArrayList<Element> list = new ArrayList<>();
        list.add(arrivalOfPatients);
        list.add(reception);
        list.add(transitionWard);
        list.add(referralLaboratory);
        list.add(registrationLaboratory);
        list.add(passingLaboratory);
        list.add(returningRecaption);

        Model model = new Model(list);

        model.simulate(time:1000000.0);
    }
}
```

Рисунок 3.9 - Код побудови моделі

```
-XX:+ShowCodeDetailsInExceptionMessages -cp /Users/andrey/Documents/Document_study/Курсов  
-----RESULTS-----  
  
Arrival of patients  
quantity = 10701  
  
Reception  
quantity = 9260  
  
Transition to the ward  
quantity = 4989  
  
Type 1: Average Duration = 27.390803918146307, Standard Deviation = 12.825190992544806  
Type 2: Average Duration = 83.39649930423063, Standard Deviation = 22.4780467524384  
  
Referral to the laboratory  
quantity = 4271  
  
Registration in the laboratory  
quantity = 4271  
  
Average Interval = 233.77584658775783, Standard Deviation = 475.40237840218094  
  
Passing tests in the laboratory  
quantity = 4271  
  
Type 3: Average Duration = 32.05972034767548, Standard Deviation = 13.66283603053498  
  
Returning to reception  
quantity = 901
```

Рисунок 3.10 - Результат виконання моделі при часі симуляції 1000000 хвилин

3.6 Верифікація моделі

Для верифікації проведемо експерименти, змінюючи кожну вхідну змінну. Кожен експеримент включає в себе 20 прогонів з часом симуляції 1000000 хвилин. Результати верифікації зображені у таблиці 3.3

Таблиця 3.3 Верифікація моделі

	Час між прибуттями в приймальне відділення	Час проходження в палату	Час з приймального відділення в лабораторію	Час обслуговування в реєстратуру лабораторії	Час проведення аналізу в лабораторії	Час, проведений хворим типу 1 у системі	Час, проведений хворим типу 2 у системі	Час, проведений хворим типу 3 у системі	Інтервал між прибуттями хворих у лабораторію
1	exp(15)	unif(5.5, 2.5)	unif(3.5, 1.5)	erl(4.5, 3)	erl(4, 2)	27,5445	85,5453	32,5150	231,1873
2	exp(30)	unif(5.5, 2.5)	unif(3.5, 1.5)	erl(4.5, 3)	erl(4, 2)	21,7779	71,1868	27,0998	403,8011
3	exp(15)	unif(20.5, 2.5)	unif(3.5, 1.5)	erl(4.5, 3)	erl(4, 2)	38,0284	93,4265	31,7746	230,5250
4	exp(15)	unif(5.5, 2.5)	unif(18.5, 1.5)	erl(4.5, 3)	erl(4, 2)	27,4479	122,1648	59,8494	230,5553
5	exp(15)	unif(5.5, 2.5)	unif(3.5, 1.5)	erl(19.5, 18)	erl(4, 2)	27,4745	99,6459	53,4777	229,3653
6	exp(15)	unif(5.5, 2.5)	unif(3.5, 1.5)	erl(4.5, 3)	erl(19, 2)	27,0149	102,2149	54,2307	233,9702

Отже, з верифікації моделі можемо зробити такі висновки:

- При збільшенні інтервалу між прибуттям зменшується час проведення хворого в системі через зменшення черг, також через це й збільшується інтервал між прибуттям хворих у лікарню.
- Час проходження в палату впливає тільки на хворих 1 і 2 типу – тому довше проводять в системі. Але цей процес йде паралельно лабораторним процесам – немає впливу.
- Час проходження до лабораторії і навпаки, не впливає на хворого 1, впливає на хворого 3, і дуже впливає на хворого 2, який проходить цей час двічі. Впливу на інтервал між прибуттям в лабораторію не видно, можливо є малий, на це може впливати відсутність черги.
- Час реєстрації на тести та час тестів, не значно впливають на час хворого 2 і 3.

4. ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ НА МОДЕЛІ

4.1 Тактичне планування

Проведемо тактичне планування для 4 величин: час проведення хворого кожного типу в системі, інтервал між прибуттям хворого в лабораторію.

Скористаємося нерівністю Чербишева (формула 4.1) для визначення кількості повторних запусків.

$$p = \frac{\sigma^2}{\varepsilon^2(1 - \beta)} \quad (4.1)$$

При $\beta=0.95$, а $\varepsilon = \frac{\sigma}{2}$, $p = 80$.

Шляхом експериментів визначенню залежності вихідних параметрів від часу моделювання визначимо перехідний період моделі. На рисунках 4.1-4.4 зображені графіки залежності вихідних параметрів від часу моделювання.



Рисунок 4.1 - Графік залежності часу проведення хворого типу 1 в системі від часу симуляції



Рисунок 4.2 - Графік залежності часу проведення хворого типу 2 в системі від часу симуляції



Рисунок 4.3 - Графік залежності часу проведення хворого типу 3 в системі від часу симуляції



Рисунок 4.4 - Інтервал між прибуттями хворих у лабораторію

Як видно з графіків, середні значення нормалізуються при часі симуляції 1500000 через велику кількість повторів, вплив більшої кількості вже незначний.

Отже, завершенням перехідного періоду моделі буде час 1500000.

Проведемо експеримент на 80 прогонах на періоді 1500000 – 2000000.

Маємо такі результати: час проведення хворого в системі – 28,0134 хвилини для першого типу з відхиленням 14,2007, 88,6894 з відхиленням 49,0969 для другого типу, 33,6736 з 13,89337 для третього типу, Інтервал між прибуттями хворих у лабораторію – 232,1874 з відхиленням 471,7783 – що позначає на великий розбіг значень інтервалів.

4.2 Стратегічне планування

Визначимо значення рівнів факторів часу між прибуттями в приймальне відділення (найбільший вплив, X_1), час проходження з приймального відділення в лабораторію або з лабораторії (інші фактори або проходять паралельно або після приходу в лабораторії, тому залежність буде нескінченно малою, X_2) в приймальне відділення інтервал між прибуттями хворих у лабораторію.

Розрахуємо значення факторів за формулами 4.2, 4.3, 4.4 та внесемо

результати у таблицю 4.1.

$$X_{i0} = \frac{X_{imax} + X_{imin}}{2} \quad (4.2)$$

$$\Delta_i = \frac{X_{imax} - X_{imin}}{2} \quad (4.3)$$

$$x_i = \frac{X_i - X_{i0}}{\Delta_i} \quad (4.4)$$

Фактор	X_{imin}	X_{imax}	X_{i0}	Δ_i
X_1	$-15\ln(s), x \in (0,s)$	$-15\ln(\epsilon)$	$-15/2\ln(\epsilon), x \in (\epsilon,1)$	$-15/2\ln(\epsilon), x \in [\epsilon,1]$
X_2	2	5	1.5	3.5

Зробимо матрицю планування (табл. 4.2). Для кожного тесту виконується 80 прогонів (розраховано в пункті 4.1). При $\epsilon=0.00001$ і $s=0.9$.

Таблиця 4.2 - Матриця планування

	X0	X1	X2	X1X2	y
1	+	+	+	+	1441.745
2	+	-	+	-	25.2825
3	+	+	-	-	83.5803
4	+	-	-	+	31.7882

Розрахуємо коефіцієнти рівняння регресії за формулою 4.5

$$b_i = \frac{\sum y_i x_{ik}}{N} \quad (4.5)$$

Результати розрахунків:

$$b_0 = 721,5342$$

$$b_1 = 696,2694$$

$$b_2 = 11,9794$$

$$b_3 = 11,9618$$

Отримаємо таке рівняння регресії (формула 4.6)

$$y_i = 721,5342 + 696,2694x_1 + 11,9794x_2 + 11,9618x_1x_2$$

Таким чином, можемо сказати, що час надходження в приймальне відділення має найбільший вплив. Фактор часу проходження з приймального відділення має теж вплив, але набагато менший.

5. ІНТЕРПРЕТАЦІЯ РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ

З верифікації моделі було зроблено такі висновки:

- При збільшенні інтервалу між прибуттям зменшується час проведення хворого в системі через зменшення черг, також через це й збільшується інтервал між прибуттям хворих у лікарню.
- Час проходження в палату впливає тільки на хворих 1 і 2 типу – тому довше проводять в системі. Але цей процес йде паралельно лабораторним процесам – немає впливу.
- Час проходження до лабораторії і навпаки, не впливає на хворого 1, впливає на хворого 3, і дуже впливає на хворого 2, який проходить цей час двічі. Впливу на інтервал між прибуттям в лабораторію не видно, можливо є малий, на це може впливати відсутність черги.
- Час реєстрації на тести та час тестів, не значно впливають на час хворого 2 і 3.

Ці тенденції є логічними, а отже можемо стверджувати, що модель валідна. В результаті тактичного планування було розраховано оптимальну кількість повторних запусків. Також, шляхом дослідження зміни показників від часу моделювання було розраховано час перехідного періоду. Таким чином, було визначено оптимальний час симуляції для дослідження системи. За отриманими результатами (час проведення хворого в системі – 28,0134 хвилини для першого типу з відхиленням 14,2007, 88,6894 з відхиленням 49,0969 для другого типу, 33,6736 з 13,89337 для третього типу, Інтервал між прибуттями хворих у лабораторію – 232,1874 з відхиленням 471,7783) можна сказати, що система працює непогано. Працездатність можна покращити за допомогою більшої кількості персоналу і пришвидшення процесу.

В результаті стратегічного планування було досліджено вплив факторів часу між прибуттями в приймальне відділення, час проходження з приймального відділення в лабораторію або з лабораторії в приймальне відділення інтервал між

прибуттями хворих у лабораторію. Було виявлено, що фактор часу надходження в приймальне відділення має найбільший вплив. Фактор часу проходження з приймального відділення має теж вплив, але набагато менший. Інші можливі фактори мають ще менший вплив. Отже, чем рідше приходять люди, тим менш коїлки до процедур, хворий проводить в системі менше часу, а інтервал приходу в лабораторію більший.

ВИСНОВКИ

У даній роботі було розроблено, створено та проаналізовано імітаційну модель приймального відділення лікарні у формалізмі МО.

У першому розділі було проведено аналіз можливих методів та засобів вирішення поставленого завдання. Було вирішено, що імітаційний метод найкраще підходить для поставленої задачі. Були визначені цілі моделювання та розроблено схематичне представлення процесу роботи моделі. Були визначені вхідні та вихідні змінні.

У другому розділі було формалізовано модель. Були визначені основні елементи системи та їх параметри, переходи, блокувальники. Було розроблено схему формалізованого представлення системи у формалізмі МО. Були представлені формули для розрахунку вихідних змінних.

У третьому розділі було визначено, що створення власної моделі МО є кращим способом автоматизації для даної задачі. Було розроблено універсальну модель МО на мові програмування Java. Використовуючи цю модель було спроектовано модель приймального відділення лікарні. Було проведено верифікацію моделі, в результаті чого було відмічено деякі залежності між вхідними і вихідними змінними. Було вирішено, що модель є валідною.

У четвертому розділі було проведено тестування моделі. В результаті тактичного планування було розраховано оптимальну кількість повторних запусків. Також, шляхом дослідження зміни показників від часу моделювання було розраховано час перехідного періоду. Таким чином, було визначено оптимальний час симуляції для дослідження системи - 1500000. Було отримано такі результати: час проведення хворого в системі – 28,0134 хвилини для першого типу з відхиленням 14,2007, 88,6894 з відхиленням 49,0969 для другого типу, 33,6736 з 13,89337 для третього типу, Інтервал між прибуттями хворих у лабораторію – 232,1874 з відхиленням 471,7783. Було виявлено, що фактор часу між прибуттями в приймальне відділення має найбільший вплив. Інші мають набагато менший вплив.

В п'ятому розділі було інтерпретовано результати тестування з 3го та 4го розділів. Було вирішено, що система працює непогано.

ДЖЕРЕЛА

1. Стеценко І. В. Моделювання систем / Інна В'ячеславівна Стеценко. – 407 с.
2. Стеценко, І. В. Моделювання систем. Курсова робота / І. В. Стеценко, О. Ю. Дифучина, А. Ю. Дифучин; КПІ ім. Ігоря Сікорського. – Київ : КПІ ім. Ігоря Сікорського, 2024. – 109 с.
3. Java [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.oracle.com/en/java/>

ДОДАТОК А

Код програми

```

--- Element.java ---
import java.util.ArrayList;
import java.util.List;

public class Element {
    private String name;
    private int id;
    private int nextId = 0;
    private List<Device> devices;
    private Transfer transfer;
    private String[][] blockers = new String[0][0];
    private String[][] changes = new String[0][0];
    private String[][] conditions = new String[0][0];
    int maxQueue = 0;

    private double tNext;
    private double tCurr;
    private int quantity;

    double deltaNow = 0;
    List<Double> type1 = new ArrayList<>();
    List<Double> type2 = new ArrayList<>();
    List<Double> type3 = new ArrayList<>();
    List<Double> entryTimes = new ArrayList<>();

    public Element(String nameOfElement, List<Double> delays, List<String> distributions, int
numberOfDevices){
        this.name = nameOfElement;
        this.tNext = 0.0;
        this.tCurr = tNext;
        this.devices = new ArrayList<>();
        for (int i = 0; i < numberOfDevices; i++) {
            this.devices.add(new Device(delays, distributions));
        }
        this.id = nextId;
        nextId++;
    }

    public void inAct(Patient patient){
    }
    public void outAct(){
        quantity++;
    }

    public void setDelay(int index, double delayDev){

```

```

Device device = devices.get(index);
device.setDelayDev(delayDev);
}

public void printResult(){
    System.out.println("\n" + this.name+ "\nquantity = "+ quantity);
    if(type1.size() > 0){
        double average = average(type1);
        double variance = variance(average, type1);
        double stdDev = Math.sqrt(variance);
        System.out.println("\nType 1: Average Duration = " + average + ", Standard Deviation = " + stdDev);
    }
    if(type2.size() > 0){
        double average = average(type2);
        double variance = variance(average, type2);
        double stdDev = Math.sqrt(variance);
        System.out.println("\nType 2: Average Duration = " + average + ", Standard Deviation = " + stdDev);
    }
    if(type3.size() > 0){
        double average = average(type3);
        double variance = variance(average, type3);
        double stdDev = Math.sqrt(variance);
        System.out.println("\nType 3: Average Duration = " + average + ", Standard Deviation = " + stdDev);
    }

    if(entryTimes.size()>0){
        List<Double> intervals = new ArrayList<>();
        for (int i = 1; i < entryTimes.size(); i++) {
            double interval = entryTimes.get(i) - entryTimes.get(i - 1);
            intervals.add(interval);
        }

        double average = average(intervals);
        double variance = variance(average, intervals);
        double stdDev = Math.sqrt(variance);
        System.out.println("\nAverage Interval = " + average + ", Standard Deviation = " + stdDev);
    }
}

public double getTcurr() {
    return tCurr;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

```

```

    }
    public void setTcurr(double tCurr) {
        this.tCurr = tCurr;
    }
    public void doStatistics(double delta){
        this.deltaNow = delta;
    }

    public List<Device> getDevices(){
        return devices;
    }

    public void setTransfer(List<Process> nextElements){
        this.transfer = new Transfer(nextElements);
    }
    public Transfer getTransfer() {
        return transfer;
    }
    public void setBlockers(String[][] blockers){
        this.blockers = blockers;
    }
    public String[][] getBlockers() {
        return blockers;
    }
    public void setChanges(String[][] changes){
        this.changes = changes;
    }
    public String[][] getChanges() {
        return changes;
    }
    public void setConditions(String[][] conditions){
        this.conditions = conditions;
    }
    public String[][] getConditions() {
        return conditions;
    }
    public void setTnext(double tNext) {
        this.tNext = tNext;
    }
    public double getTnext() {
        return this.tNext;
    }
    }

    public void updateTnext(){
        this.tNext = devices.stream()
            .mapToDouble(Device::getTNext)
            .min()
            .orElse(this.tNext);
    }

    public double average(List<Double> list){
        return list.stream().mapToDouble(Double::doubleValue).average().orElse(0);
    }

```



```

    }

    public double variance(double average, List<Double> list){
        return list.stream()
            .mapToDouble(x -> Math.pow(x - average, 2))
            .average()
            .orElse(0);
    }
}

```

--- FunRand.java ---

```

public class FunRand {
    public static double Exp(double timeMean) {
        double a = 0;

        while (a == 0) {
            a = Math.random();
        }

        a = -timeMean * Math.log(a);

        return a;
    }

    public static double Unif(double timeMin, double timeMax) {
        double a = 0;

        while (a == 0) {
            a = Math.random();
        }

        a = timeMin + a * (timeMax - timeMin);

        return a;
    }

    public static double Erl(double timeMean, double k) {
        double a = 0;

        for (int i = 0; i < k; i++) {
            double u = 0;
            while (u == 0) {
                u = Math.random();
            }

            a += Math.log(u);
        }

        a = -1 / timeMean * a;
    }
}

```

```

        return a;
    }
}

```

--- Process.java ---

```

import java.util.ArrayList;
import java.util.List;

```

```

public class Process extends Element {

```

```

    private QueueP queue = new QueueP();

```

```

    public Process(String nameOfElement, List<Double> delays, List<String> distributions, int
numberOfDevices) {
        super(nameOfElement, delays, distributions, numberOfDevices);
    }

```

```

    @Override

```

```

    public void inAct(Patient patient) {
        String name = super.getName();
        int type = patient.getType();
        if(name == "Registration in the laboratory") {
            entryTimes.add(super.getTcurr());
        }
        for (Device device : super.getDevices()) {
            device.setTcurr(super.getTcurr());
            if (device.getState() == 0) {
                int delayNumber = device.getDelayNumber();
                if(delayNumber == 3){
                    device.inAct(patient, type-1);
                } else {
                    device.inAct(patient, 0);
                }
            }
            super.updateTnext();
            return;
        }
    }
}

```

```

        queue.add(patient);
        super.maxQueue = queue.length()>super.maxQueue ? queue.length() : super.maxQueue;
    }
}

```

```

    @Override

```

```

    public void outAct( ) {
        List<Patient> outTasks = new ArrayList<>();
        for (Device device : super.getDevices()) {
            device.setTcurr(super.getTcurr());
            if (device.getTNext() == super.getTcurr()) {
                Patient task = device.outAct();
                if(task instanceof Patient && task != null) {
                    outTasks.add(task);
                }
            }
        }
    }
}

```

```

    }
    }
}
if (outTasks.size() > 0){
    for (Patient task : outTasks) {
        super.outAct();
        Transfer transfer = super.getTransfer();
        if(transfer != null){
            String next = transfer.goNext(this, super.getTcurr(), task);
            if((next != "blocked") && queue.length() > 0){
                Patient patient = this.queue.remove();
                int type = patient.getType();
                for (Device device : super.getDevices()) {
                    device.setTcurr(super.getTcurr());
                    if (device.getState() == 0) {
                        int delayNumber = device.getDelayNumber();
                        if(delayNumber == 3){
                            device.inAct(patient, type-1);
                        } else {
                            device.inAct(patient, 0);
                        }
                    }
                    super.updateTnext();
                    return;
                }
            }
        }
        if(next == null){
            this.addDuration(task);
        }
    }
    if(transfer == null){
        this.addDuration(task);
    }
}
}
for (Device device : super.getDevices()) {
    if (device.getState() == 0) {
        device.setTNext(Double.MAX_VALUE);
    }
}
super.updateTnext();
}

```

```

@Override
public void doStatistics(double delta){
}

```

```

public void addDuration(Patient patient) {
    int type = patient.getType();
    int flag = patient.getFlag();
    patient.setExit(super.getTcurr());
}

```

```

        if (type == 1) {
            if(flag == 0) {
                super.type1.add(patient.getDuration());
            } else {
                super.type2.add(patient.getDuration());
            }
        } else if (type == 3) {
            super.type3.add(patient.getDuration());
        }
    }
}

}

--- Create.java ---
import java.util.List;

public class Create extends Element{
    public Create(String nameOfElement, List<Double> delays, List<String> distributions, int
numberOfDevices) {
        super(nameOfElement, delays, distributions, numberOfDevices);
        for (Device device : super.getDevices()) {
            device.setTNext(0);
        }
    }

    @Override
    public void outAct( ) {
        super.outAct();
        Time time = new Time(super.getTcurr());
        double hours = time.getHours();
        double p1,p2,p3;
        if(hours >= 7 && hours <= 10 ){
            p1 = 0.9;
            p2 = 0.1;
            p3 = 0;
        } else {
            p1 = 0.5;
            p2 = 0.1;
            p3 = 0.4;
        }

        Patient newPatient = generatePatient(p1, p2, p3);
        newPatient.setArrival(super.getTcurr());
        double delay = 0;
        for (Device device : super.getDevices()) {
            device.setPatient(newPatient);
            delay = device.getDelayAverage(0);
        }
        Transfer transfer = super.getTransfer();
        String next = transfer.goNext(this, super.getTcurr(), newPatient);
    }
}

```

```

        if((next == "blocked")){
            super.setTnext(super.getTcurr() + time.getNightDelay());
        } else {
            super.setTnext(super.getTcurr() + delay);
        }
    }

}

public Patient generatePatient(double p1, double p2, double p3) {
    int type = 0;
    double rand = Math.random();

    if (rand < p1) {
        type = 1;
    } else if (rand < p1 + p2) {
        type = 2;
    } else {
        type = 3;
    }

    Patient newPatient = new Patient(type, super.getTcurr());

    return newPatient;
}
}

--- Time.java ---
public class Time {
    private double minutesOfDay;
    private double hours, minutes, seconds;

    public Time(double minutes) {
        minutesOfDay = minutes;
    }

    public String toHHMMSS() {
        hours = Math.floor((minutesOfDay / 60) % 24);
        minutes = Math.floor(minutesOfDay % 60);
        seconds = ((minutesOfDay % 60) - minutes) * 60 / 100;
        return String.format("%02.0f:%02.0f:%02.0f", hours, minutes, seconds);
    }

    public void setMinutes(double minutes) {
        minutesOfDay = minutes;
    }

    public double getHours() {
        return (minutesOfDay / 60) % 24;
    }

    public double getNightDelay() {

```

```

double minutesUntilSeven = 0;

if (hours < 7) {
    minutesUntilSeven = (7 * 60) - (hours * 60 + minutes);
} else {
    minutesUntilSeven = (24 * 60) - (hours * 60 + minutes) + (7 * 60);
}

return minutesUntilSeven;
}
}

```

--- Device.java ---

```

import java.util.List;

public class Device {
    private List<Double> delaysAverage;
    private List<String> distributions;
    private double delayDev;

    private double tNext = Double.MAX_VALUE;
    private double tCurr = 0;

    private int state;
    private Patient patient;

    public Device(List<Double> delays, List<String> distributions) {
        this.delaysAverage = delays;
        this.distributions = distributions;
        this.state = 0;
    }

    public double getDelayAverage(int index) {
        double delay = delaysAverage.get(index);
        String distribution = distributions.get(index);

        switch (distribution.toLowerCase()) {
            case "exp":
                return FunRand.Exp(delay);
            case "erl":
                return FunRand.Erl(delay, this.delayDev);
            case "unif":
                return FunRand.Unif(delay, this.delayDev);
            default:
                return delay;
        }
    }

    public void inAct(Patient patient, int index) {
        state = 1;
    }
}

```

```

        this.patient = patient;
        this.tNext = tCurr + this.getDelayAverage(index);
    }

    public Patient outAct() {
        state = 0;
        Patient result = this.patient;
        this.patient = null;
        return result;
    }

    public void setTNext(double tNext) {
        this.tNext = tNext;
    }
    public double getTNext() {
        return tNext;
    }

    public void setTcurr(double tCurr) {
        this.tCurr = tCurr;
    }

    public int getState() {
        return state;
    }

    public void setDelayDev(double delayDev) {
        this.delayDev = delayDev;
    }

    public void setPatient(Patient patient){
        this.patient = patient;
    }
    public Boolean isPatient() {
        return this.patient != null;
    }

    public void addWaiting(int index) {
        state = 1;
        this.patient = null;
        this.tNext = tCurr + this.getDelayAverage(index);
    }

    public int getDelayNumber() {
        return delaysAverage.size();
    }
}

```

```

--- Transfer.java ---
import java.util.List;

```

```

public class Transfer {
    private List<Process> nextElements;

    public Transfer(List<Process> nextElements){
        this.nextElements = nextElements;
    }
    public String goNext(Process current, double tCurr, Patient patient){
        int index = this.getIndex(current, tCurr, patient);
        if(index >= 0){
            Process nextElement = nextElements.get(index);
            nextElement.inAct(patient);
            return "free";
        } else if (index == -1){
            return "blocked";
        } else {
            return null;
        }
    }
    public String goNext(Create current, double tCurr, Patient patient){
        Time time = new Time(tCurr);
        double hours = time.getHours();

        if(hours >= 7 && hours <= 16){
            Process nextElement = nextElements.get(0);
            nextElement.inAct(patient);
            return "free";
        } else {
            return "blocked";
        }
    }
    public int getIndex(Process current, double tCurr, Patient patient) {

        Time time = new Time(tCurr);
        double hours = time.getHours();
        int type = patient.getType();
        int flag = patient.getFlag();
        int index = 0;
        int currentElement = 0;

        for(Process element : nextElements ){
            String[][] conditions = element.getConditions();
            String[][] blockers = element.getBlockers();
            String[][] changes = element.getChanges();

            if(blockers.length != 0) {
                for (String[] condition : blockers) {
                    boolean flagCondition = true;
                    boolean timeCondition = true;

                    for (int i = 0; i < condition.length; ) {
                        String key = condition[i];

```



```

        if ("flag".equals(key)) {
            int conditionFlag = Integer.parseInt(condition[i + 1]);
            flagCondition = (flag == conditionFlag);
            i += 2;
        } else if ("time".equals(key)) {
            double start = Double.parseDouble(condition[i + 1]);
            double end = Double.parseDouble(condition[i + 2]);
            timeCondition = (hours >= start && hours <= end);
            i += 3;
        } else {
            i++;
        }
    }
    if (flagCondition && timeCondition) {
        index = 0;
        break;
    }
    return -1;
}
}

if(conditions.length != 0) {

    for (String[] condition : conditions) {
        boolean typeCondition = true;

        for (int i = 0; i < condition.length; ) {
            String key = condition[i];

            if ("type".equals(key)) {
                int conditionFlag = Integer.parseInt(condition[i + 1]);
                typeCondition = (type == conditionFlag);
                i += 2;
            } else if ("!type".equals(key)) {
                int conditionFlag = Integer.parseInt(condition[i + 1]);
                typeCondition = (type != conditionFlag);
                i += 2;
            } else {
                i++;
            }
        }
        if (typeCondition) {
            index = currentElement;
            break;
        }
        if(nextElements.size() == 1) {
            return -2;
        }
    }
}
}

```

```

        if(changes.length != 0) {
            for (String[] change : changes) {
                for (int i = 0; i < change.length; ) {
                    String key = change[i];

                    if ("type".equals(key)) {
                        int changeValue = Integer.parseInt(change[i + 1]);
                        patient.setType(changeValue);
                        i += 2;
                    } else if ("flag".equals(key)) {
                        int changeValue = Integer.parseInt(change[i + 1]);
                        patient.setFlag(changeValue);
                        i += 2;
                    } else {
                        i++;
                    }
                }
            }
        }

        currentElement++;
    }
    return index;
}

}

public Process getNext(){
    return nextElements.get(0);
}

}

```

--- QueueP.java ---

```

import java.util.ArrayList;
import java.util.List;

public class QueueP {
    private List<Patient> patients;
    private int currentLength;
    private String status = "empty";

    public QueueP() {
        currentLength=0;
        patients = new ArrayList<>();
    }

    public String getStatus() {
        currentLength = patients.size();
        if(currentLength == 0) {
            status = "empty";
        } else {

```

```

        status = "notEmpty";
    }

    return status;
}

public void add(Patient patient) {
    patients.add(patient);
    currentLength = patients.size();
}

public Patient remove() {
    return patients.remove(0);
}

public int length(){
    currentLength = patients.size();
    return currentLength;
}
}

```

--- App.java ---

```

import java.util.ArrayList;
import java.util.Arrays;

```

```

public class App {
    public static void main(String[] args){

```

```

        Create arrivalOfPatients = new Create("Arrival of patients", Arrays.asList(15.0), Arrays.asList("exp"), 1);

```

```

        Process reception = new Process("Reception", Arrays.asList(15.0, 40.0, 20.0), Arrays.asList("", "", ""),
2);

```

```

        Process transitionWard = new Process("Transition to the ward", Arrays.asList(3.0, 3.0),
Arrays.asList("unif", ""), 3);
        transitionWard.setDelay(0, 8.0);

```

```

        Process referralLaboratory = new Process("Referral to the laboratory", Arrays.asList(2.0),
Arrays.asList("unif"), 1);
        referralLaboratory.setDelay(0, 5.0);

```

```

        Process registrationLaboratory = new Process("Registration in the laboratory", Arrays.asList(4.5),
Arrays.asList("erl"), 1);
        registrationLaboratory.setDelay(0, 3.0);

```

```

        Process passingLaboratory = new Process("Passing tests in the laboratory", Arrays.asList(4.0),
Arrays.asList("erl"), 2);
        passingLaboratory.setDelay(0, 2.0);

```

```

        Process returningRecaption= new Process("Returning to reception", Arrays.asList(2.0),
Arrays.asList("unif"), 1);
        passingLaboratory.setDelay(0, 5.0);

```

```

arrivalOfPatients.setTransfer(Arrays.asList(reception));

reception.setBlockers(new String[][]{{"flag", "1", "time", "7", "16"}, {"time", "7", "17"}});
reception.setTransfer(Arrays.asList(transitionWard, referralLaboratory));

transitionWard.setConditions(new String[][]{{"type", "1"}});

referralLaboratory.setConditions(new String[][]{{"!type", "1"}});
referralLaboratory.setTransfer(Arrays.asList(registrationLaboratory));

registrationLaboratory.setTransfer(Arrays.asList(passingLaboratory));

passingLaboratory.setTransfer(Arrays.asList(returningRecaption));

returningRecaption.setConditions(new String[][]{{"type", "2"}});
returningRecaption.setTransfer(Arrays.asList(reception));
returningRecaption.setChanges(new String[][]{{"type", "1"}, {"flag", "1"}});

ArrayList<Element> list = new ArrayList<>();
list.add(arrivalOfPatients);
list.add(reception);
list.add(transitionWard);
list.add(referralLaboratory);
list.add(registrationLaboratory);
list.add(passingLaboratory);
list.add(returningRecaption);

Model model = new Model(list);

model.simulate(1000000.0);
}
}

```

--- Model.java ---

```

import java.util.ArrayList;

public class Model {
    private double tNext = 0;
    private double tCurr = 0;
    private ArrayList<Element> list = new ArrayList<>();

    public Model(ArrayList<Element> elements) {
        list = elements;
    }

    public void simulate(double time) {
        while (this.tCurr < time) {
            this.tNext = Double.MAX_VALUE;
            for (Element e : list) {
                if ((e.getTnext() < this.tNext)) {
                    this.tNext = e.getTnext();
                }
            }
        }
    }
}

```

```

    }
    }
    for (Element e : list) {
        e.doStatistics(tNext - tCurr);
    }
    this.tCurr = this.tNext;
    for (Element e : list) {
        e.setTcurr(tCurr);
    }
    for (Element e : list) {
        if (e.getTnext() == this.tCurr) {
            e.outAct();
        }
    }
}
printResult();
}

public void printResult() {
    System.out.println("\n-----RESULTS-----");
    for (Element e : list) {
        e.printResult();
    }
}
}

```

--- Patient.java ---

```

public class Patient {
    private int type;

    private double arrivalTime;
    private double exit;
    private double duration;

    private int isPassed;

    public Patient(int type, double time) {
        this.type = type;
        arrivalTime = time;
        duration = 0;
        exit = 0;
        isPassed = 0;
    }

    public void setType(int type) {
        this.type = type;
    }
    public int getType() {
        return type;
    }
    public double getDuration() {

```

```

        return duration;
    }
    public void setExit(double exit) {
        this.exit = exit;
        this.duration+= exit - this.arrivalTime;
    }
    public void setArrival(double arrivalTime) {
        this.arrivalTime = arrivalTime;
    }

    public String getPatient() {
        String patiet = "Type: " + type + ", arrived at:" + arrivalTime + ", exit at:" + exit + ", duration: " +
duration;
        return patiet;
    }

    public int getFlag() {
        return isPassed;
    }
    public void setFlag(int isPassed) {
        this.isPassed = isPassed;
    }
}

```