НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»  
Кафедра обчислювальної техніки

**Курсовий проект**

з дисципліни «Системне програмне забезпечення»

на тему: «Планування розподілу завдань

у багатопроцесорній системі з топологією «Матриця процесорів 3х3 - неоднорідна»

Студента 4-го курсу групи ІО-43

напряму підготовки «Комп’ютерна інженерія»

Крута В. В.

Керівник доктор т.н., професор Сімоненко В. П.

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів:\_\_\_\_\_\_\_\_\_ Оцінка: ECTS\_\_\_\_\_

\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ — 2017

ОПИС АЛЬБОМУ

ТЕХНІЧНЕ ЗАВДАННЯ

1 НАЙМЕНУВАННЯ І СФЕРА ЗАСТОСУВАННЯ

Розроблюється програма, що має розв'язувати задачу статичного планування для багатопроцесорної системи із топологією “Матриця процесорів 3х3 - неоднорідна”.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання, отримане на курсову роботу з дисципліни “Системне програмне забезпечення” для 4 курсу навчального напряму “Комп'ютерна інженерія” НТУУ “КПІ імені І. Сікорського”.

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Програма має виконувати статичне планування і завантаження задач на процесори.

4 ВИМОГИ ДО ДОКУМЕНТАЦІЇ

Має бути розроблена наступна документація:

- пояснювальна записка, що включає опис програми і приклад її виконання;

- лістинг програми.

5 ЕТАПИ ТА СТАДІЇ РОЗРОБКИ

1. Аналіз відомих рішень.

2. Вибір алгоритму планування.

3. Розробка програми.

4. Розробка документації.

5. Захист курсового проекту.

ПОЯСНЮВАЛЬНА ЗАПИСКА

ЗМІСТ

[ВСТУП 2](#_Toc501619494)

[РОЗДІЛ 1. ОПИС ТОПОЛОГІЇ СИСТЕМИ 3](#_Toc501619495)

[РОЗДІЛ 2. ВИБІР АЛГОРИТМУ ДЛЯ ПЛАНУВАННЯ 4](#_Toc501619496)

[2.1. Класифікація алгоритмів 4](#_Toc501619497)

[2.2. Алгоритм MH 5](#_Toc501619498)

[РОЗДІЛ 3. ПРИКЛАД РОБОТИ ПРОГРАМИ 7](#_Toc501619499)

[ВИСНОВКИ 11](#_Toc501619500)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 12](#_Toc501619501)

[Додаток А 13](#_Toc501619502)

ВСТУП

Ця робота присвячена розв'язанню задачі статичного планування для багатопроцесорної системи. Мета статичного планування — складання розкладу виконання задач і розподілу їх по процесорах так, щоб мінімізувати довжину планування. В класичному підході, який також називається списковим плануванням, основною ідеєю є упорядкування списку вузлів шляхом присвоєння їм пріоритетів, закріплення завдань по процесорах, а потім повторення наступних двох кроків з урахуванням виконаного призначення до отримання плану розв'язання [1].

У роботі розроблюється програма-планувальник для системи заданої топології, а саме топології «матриця процесорів 3х3 - неоднорідна».

Не можна сказати, який із існуючих (і тих, що будуть розроблені в майбутньому) алгоритмів статичного планування є оптимальним. На різних топологіях і різних типах графів задач порівняна ефективність алгоритмів є різною, кожен має свої алгоритми й недоліки, які слід аналізувати.

РОЗДІЛ 1. ОПИС ТОПОЛОГІЇ СИСТЕМИ

Топологія “матриця процесорів 3х3 - неоднорідна” відрізняється від топології “матриця процесорів 3х3” тим, що що потужність процесорів не однакова. Один процессор може виконувати певну задачу у декілька разів швидше за інший. Ця топологія є двонаправленою, тобто кожен процесор передає дані будь-якому процесору та отримує дані від будь-якого процесора, з якими він безпосередньо з‘єднаний. Таким чином, діаметр топології дорівнює √P - 1, де P — число процесорів. У нашому випадку – 4.

Приклад системи топології “матриця процесорів 3х3 - неоднорідна” показано на рисунку 1.1.

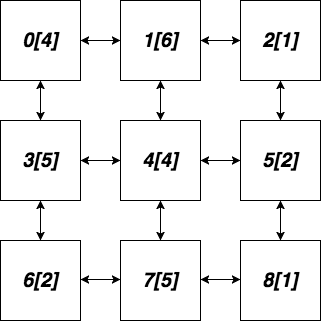


Рис. 1.1. Система топології “матриця процесорів 3х3 - неоднорідна”

РОЗДІЛ 2. ВИБІР АЛГОРИТМУ ДЛЯ ПЛАНУВАННЯ

2.1. Класифікація алгоритмів

Існують алгоритми для різних типів графів задач. У цій роботі нас цікавлять насамперед ті з них, що працюють із задачами довільної ваги, враховуючи взаємодію (пересилання даних між ними). Серед таких можна виділити класи UNC (unbounded number of clusters) і BNP (bounded number of processors).

Вказані два класи алгоритмів відрізняються тим, що робота UNC-алгоритмів є принципово двоетапною: етап кластеризації задач (кількість кластерів необмежена), потім етап встановлення відповідності “кластер — процесор”; BNP-алгоритми, розв'язують задачу планування, одразу враховуючи обмежену кількоість процесорів. З числа BNP виділяється клас алгоритмів APN (arbitrary processor network), які відрізняються тим, що враховують топологію системи, для якої виконується планування. Таким чином, алгоритми APN працюють навіть для неповнозв’язних топологій.

Оскільки задана за варіантом топологія “матриця процесорів 3х3 - неоднорідна” є неповнозв'язною, оберемо для реалізації один з алгоритмів класу APN. Клас APN включає, зокрема, такі алгоритми: BU (Bottom Up), BSA (Bubble Scheduling and Allocation), MH (Mapping Heuristic), DLS (Dynamic Level Scheduling).

Алгоритм BU (Bottom Up) починає з планування критичного шляху графу (тобто найдовшого шляху з початкової вершини до кінцевої) на один процесор. Решта задач розподіляється по процесорах з міркувань рівномірності навантаження на процесори, а також на лінії зв'язку.

Алгоритм BSA (Bubble Scheduling and Allocation) на першому етапі виконання завантажує всі задачі на найбільш досяжний процесор (тобто той, що має найбільше сусідів), потім обходить його сусідів і переміщує на них задачі, якщо це дає виграш у часі виконання, після чого повторює ітерацію для одного з цих сусідів, і так до обходу всієї системи.

Алгоритми MH (Mapping Heuristic) і DLS (Dynamic Level Scheduling) є доволі подібними, основною відмінністю є формула пріоритету задачі: у випадку MH це статичний b-рівень, а у випадку DLS — динамічний рівень, що визначається як різниця між статичним b-рівнем і найбільш раннім часом початку виконання задачі.

Видається доречним віддати перевагу алгоритму MH порівняно з алгоритмом DLS, оскільки у роботі DLS можлива ситуація, коли він планує більш важливі вузли відносно пізно, віддаючи перевагу вузлам з більш раннім часом початку. Це потенційно призводить до неоптимальності планування, яка у топології з невеликою зв'язністю (зокрема у “матриці процесорів”) може призвести до дуже суттєвого збільшення довжини планування. Алгоритм MH, натомість, демонструє більш стабільну роботу.

2.2. Алгоритм MH

Алгоритм MH складається з таких кроків:

1) Обчислити статичний b-рівень (sl) кожної вершини графу задач. Під статичним b-рівнем мається на увазі довжина найдовшого можливого шляху від вершини, що розглядається, до кінцевої вершини, з урахуванням самих цих вершин, але без урахування пересилань. sl визначає пріоритет вузла. Якщо два вузли мають однаковий sl, пріоритетнішим вважається вузол із більшим числом вузлів-нащадків. Якщо між вузлами “нічия” за цим параметром теж, конфлікт розв'язується випадковим чином. Черга готових вузлів на цьому етапі включає тільки початкові вузли. Ця черга сортована за спаданням пріоритетів.

2) Для чергової задачі визначається процесор, на якому час її початку буде найбільш раннім. Для визначення часу початку аналізується час звільнення процесора, час завершення всіх предків задачі і пересилання даних від цих предків з урахуванням топології системи. Задача планується на цей процесор.

3) Вузли, всі предки яких вже заплановані, додаються до черги готових.

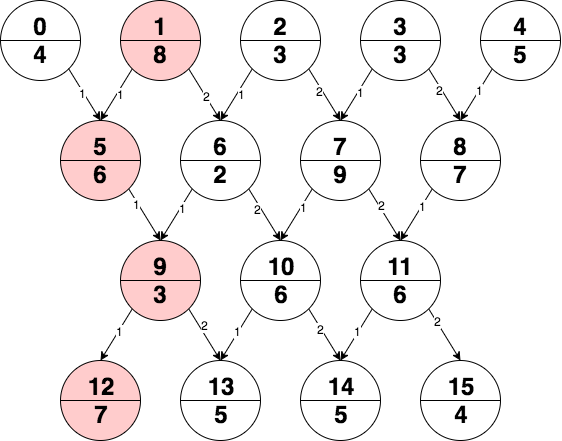
4) Повернення до кроку 2, поки всі задачі не будуть розподілені по процесорах [3].

РОЗДІЛ 3. ПРИКЛАД РОБОТИ ПРОГРАМИ

Виконаємо планування графа задач, який показано на рисунку 3.1, на систему, схему якої показано на рисунку 1.1 в розділі 1.

У графі задач визначимо найдовший шлях з початкового вузла до кінцевого без урахування ваги пересилань. Цей шлях (довжина якого складає 24) на графі виділено кольором. Довжина цього шляху дозволяє оцінити мінімальний час планування — цей мінімум, утім, не обов'язково може бути досягнутий.

Рис. 3.1. Граф задач



На рисунку 3.2 графік Ганта, що представляє результати планування.

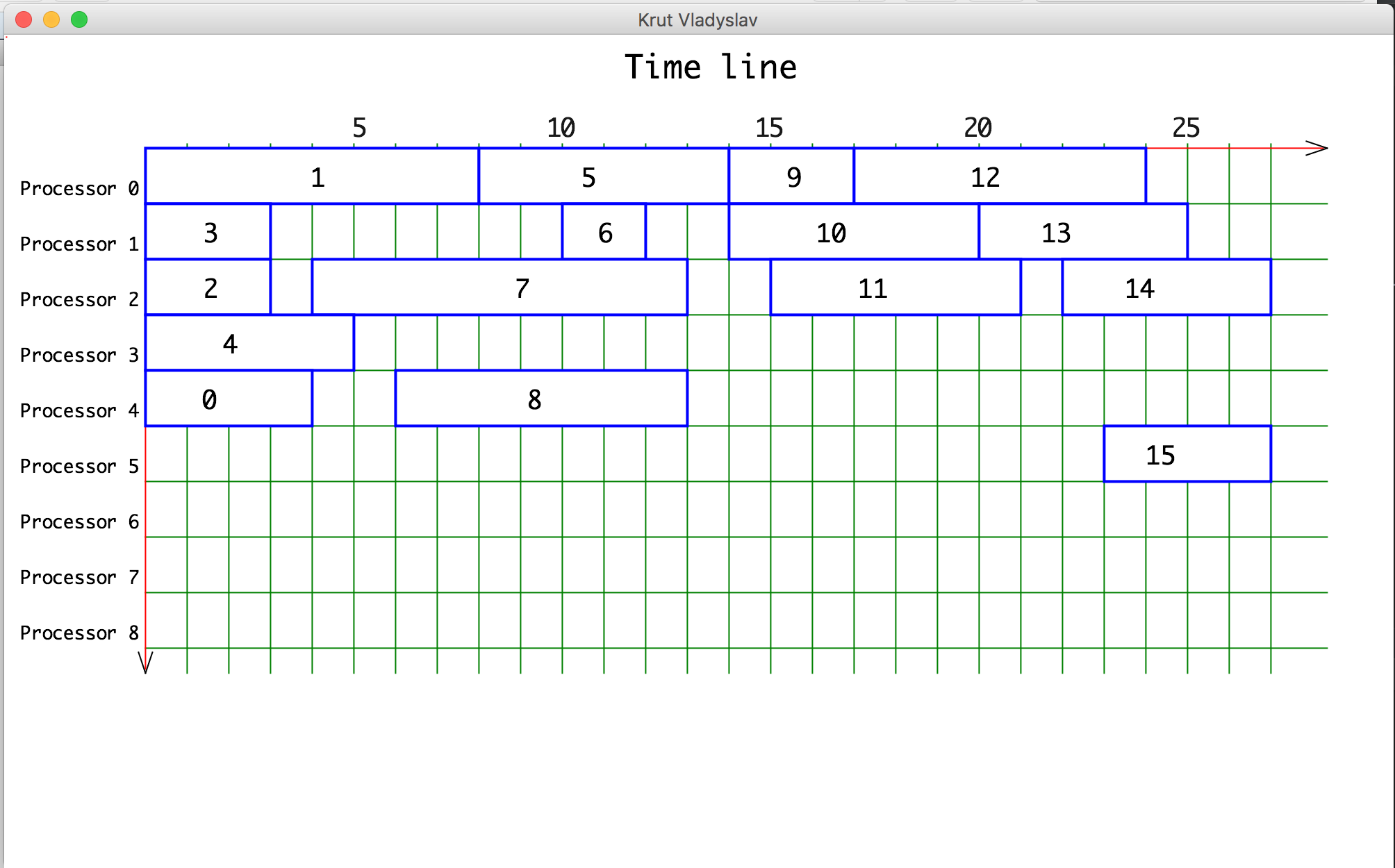


Рис. 3.2. Результат планування

Задачі плануються у порядку відповідно до статичного b-рівня, як показано у таблиці 3.1.

Пересилання даних між процесорами показано в таблиці 3.2.

Таблиця 3.1. Порядок планування задач

|  |  |
| --- | --- |
| Номер задачі | sl |
| 1 | 24 |
| 3 | 23 |
| 2 | 23 |
| 4 | 23 |
| 7 | 20 |
| 0 | 20 |
| 8 | 18 |
| 5 | 16 |
| 6 | 13 |
| 10 | 11 |
| 11 | 11 |
| 9 | 10 |
| 12 | 7 |
| 13 | 5 |
| 14 | 5 |
| 15 | 4 |

Таблиця 3.2. Міжпроцесорні пересилання (у порядку планування)

|  |  |  |  |
| --- | --- | --- | --- |
| Лінк | Часовий інтервал | Задача-відправник | Задача-отримувач |
| P1-P2 | 4 | 3 | 7 |
| P1-P4 | 5 | 3 | 8 |
| P3-P4 | 6 | 4 | 8 |
| P4-P1 | 5 | 0 | 5 |
| P1-P0 | 6 | 0 | 5 |
| P0-P1 | 10 | 1 | 6 |
| P2-P1 | 4 | 2 | 6 |
| P2-P1 | 14 | 7 | 10 |
| P4-P1 | 14 | 8 | 11 |
| P1-P2 | 15 | 8 | 11 |
| P1-P0 | 13 | 6 | 9 |
| P0-P1 | 19 | 9 | 13 |
| P1-P2 | 22 | 10 | 14 |
| P2-P5 | 23 | 11 | 15 |

Через те, що ширина графу задач дорiвнює 5 вершин, ми могли очікувати, що задіяно буде всього лише 5 процесорів. Задачі такого типу не дуже ефективно виконувати на 9 – процесорній системі. Проте в кінці програми стається випадок, коли процесор “2” мусить передати результат задачі “11” вільному процесору для виконання задачі “15”. Процесор “2” – кутовий, тобто він має всього два зв’язки: з процесорами “1” та “5”, при чому процесор “1” зайнятий виконанням іншої задачі. Це саме той випадок, коли на перший погляд “зайвий” процесор стає у нагоді і дозволяє значно пришвидшити обчислення.

Як видно, довжина планування (27 тактів) близька до оптимальної (24). Це цілком вдалий результат, який досягається саме завдяки додатковому процесору, який решту часу виконання простоює.

Можна зробити висновок, що на даній топології на цьому тестовому прикладі алгоритм планування MH працює нормально і забезпечує цілком прийнятні результати.

ВИСНОВКИ

В ході виконання курсового проекту було розглянуто постановку задачі статичного планування і, в рамках підходу спискового планування, різні алгоритми розв'язання цієї задачі.

Згідно з обраним алгоритмом MH розроблено програму, що здійснює планування для заданої за варіантом топології “матриця процесорів 3х3 - неоднорідна”.

Виконано тестування програми та аналіз результатів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1) Методичні вказівки до виконання курсової роботи для студентів напрямку підготовки 6.050102 «Комп’ютерна інженерія» професійного спрямування «Комп’ютерні системи та мережі» денної та заочної форми навчання/ укладачі: В.П. Сімоненко, А.В. Сімоненко – К.: НТУУ «КПІ» ,2012. - 81с.

2) Таненбаум Э., Уэзеролл Д. Т18 Компьютерные сети. 5-е изд. — СПб.: Питер, 2012. — 960 с.

3) El-Rewini H., Lewis T. G. Scheduling parallel program tasks onto arbitrary target machines. Journal of parallel and distributed Computing 9, 1990, 138 –153.

Додаток А

Лістинг основної частини програми

package com.gantt.sample;  
  
import com.Link;  
import com.Graph;  
import com.Processor;  
import javafx.scene.text.FontWeight;  
import com.Transmission;  
import com.Task;  
import com.APN\_MH;  
import com.gantt.entity.Model;  
import javafx.application.Application;  
import javafx.scene.Group;  
import javafx.scene.Scene;  
import javafx.scene.control.ScrollPane;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.\*;  
import javafx.scene.text.Font;  
import javafx.scene.text.Text;  
import javafx.stage.Stage;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Main extends Application {  
 Group root;  
 Model model;  
 int PROC\_NUMBER;  
 double START\_X = 100;  
 double START\_Y = 80;  
 double END\_X = 50;  
 double END\_Y = 50;  
 double TASK\_HEIGHT = 10;  
 double TASK\_WEIGHT = 15;  
 double RATE\_H = 2;  
 double RATE\_V = 2;  
 double INTERVAL = 10;  
 int FONT\_HEIGHT = 18;  
 double MAX\_TIME = 0;  
  
 @Override  
 public void init() throws Exception {  
 super.init();  
 //set processors  
 PROC\_NUMBER = 9;  
 List<Processor> processors = new ArrayList<Processor>();  
 List<Link> links = new ArrayList<Link>();  
  
 processors.add(new Processor(4));  
 processors.add(new Processor(6));  
 processors.add(new Processor(1));  
 processors.add(new Processor(5));  
 processors.add(new Processor(4));  
 processors.add(new Processor(2));  
 processors.add(new Processor(2));  
 processors.add(new Processor(5));  
 processors.add(new Processor(1));  
  
 //set links  
  
 addLane(processors, links, "Link 0-1", 0, 1, 1);  
 addLane(processors, links, "Link 1-0", 1, 0, 1);  
 addLane(processors, links, "Link 0-3", 0, 3, 1);  
 addLane(processors, links, "Link 3-0", 3, 0, 1);  
  
 addLane(processors, links, "Link 1-2", 1, 2, 1);  
 addLane(processors, links, "Link 2-1", 2, 1, 1);  
 addLane(processors, links, "Link 1-4", 1, 4, 1);  
 addLane(processors, links, "Link 4-1", 4, 1, 1);  
  
 addLane(processors, links, "Link 2-5", 2, 5, 1);  
 addLane(processors, links, "Link 5-2", 5, 2, 1);  
  
 addLane(processors, links, "Link 3-4", 3, 4, 1);  
 addLane(processors, links, "Link 4-3", 4, 3, 1);  
 addLane(processors, links, "Link 3-6", 3, 6, 1);  
 addLane(processors, links, "Link 6-3", 6, 3, 1);  
  
 addLane(processors, links, "Link 4-5", 4, 5, 1);  
 addLane(processors, links, "Link 5-4", 5, 4, 1);  
 addLane(processors, links, "Link 4-7", 4, 7, 1);  
 addLane(processors, links, "Link 7-4", 7, 4, 1);  
  
 addLane(processors, links, "Link 5-8", 5, 8, 1);  
 addLane(processors, links, "Link 8-5", 8, 5, 1);  
  
 addLane(processors, links, "Link 6-7", 6, 7, 1);  
 addLane(processors, links, "Link 7-6", 7, 6, 1);  
  
 addLane(processors, links, "Link 7-8", 7, 8, 1);  
 addLane(processors, links, "Link 8-7", 8, 7, 1);  
  
  
 Graph graph = new Graph(processors, links);  
  
 //set task nodes  
 int tNumber = 16;  
 int[] weights = { 4, 8, 3, 3, 5,  
 6, 2, 9, 7,  
 3, 6, 6,  
 7, 5, 5, 4 };  
 //set task transmissions  
 int[][] matrix = {  
 {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0},  
  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0},  
  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2},  
  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}  
 };  
 ArrayList<Task> tasks = defineGraph(matrix, weights);  
  
 //schedule  
 APN\_MH APNMh = new APN\_MH(graph);  
 int[][] plan = APNMh.schedule(tasks);  
  
 //visualize  
 model = new Model(PROC\_NUMBER);  
 for (int i = 0; i < tNumber; i++) {  
 model.getProcessor(plan[0][i]).addTask(new com.gantt.entity.Task(i, plan[1][i], plan[2][i]));  
 }  
  
 }  
  
 public static void main(String[] args) {  
 launch(args);  
 }  
  
 private static void addLane(List<Processor> processors, List<Link> links, String laneId, int sourceLocNo, int destLocNo,  
 int duration) {  
 Link lane = new Link(laneId, processors.get(sourceLocNo), processors.get(destLocNo), duration);  
 links.add(lane);  
 }  
  
 private static ArrayList<Task> defineGraph(int[][] matrix, int[] weights) {  
 ArrayList<Task> tasks = new ArrayList<>();  
 for (int i = 0; i < matrix.length; i++) {  
 Task task = new Task(weights[i], i);  
 tasks.add(task);  
 }  
 for (int i = 0; i < matrix.length; i++) {  
 for (int j = 0; j < matrix.length; j++) {  
 if (matrix[i][j] != 0) {  
 tasks.get(i).addChild(new Transmission(tasks.get(j), matrix[i][j]));  
 tasks.get(j).addParent(new Transmission(tasks.get(i), matrix[i][j]));  
 }  
 }  
 }  
 return tasks;  
 }  
  
 public void initiateData() {  
 drawAxes();  
 drawTasks();  
  
 }  
  
 @Override  
 public void start(Stage primaryStage) throws Exception {  
 ScrollPane sp = new ScrollPane();  
 root = new Group();  
 primaryStage.setTitle("Krut Vladyslav");  
 Scene scene = new Scene(sp, 1000, 600);  
  
 scene.getStylesheets().add("com/gantt/resources/textstyle.css");  
 primaryStage.setScene(scene);  
  
  
 Line line = new Line();  
 line.setStartX(0);  
 line.setStartY(0);  
 line.setEndX(0);  
 line.setEndY(0);  
 line.setStyle("-fx-stroke: red;");  
 root.getChildren().add(line);  
  
 sp.setLayoutX(0);  
 sp.setFitToWidth(true);  
 sp.setContent(root);  
 sp.setHbarPolicy(ScrollPane.ScrollBarPolicy.AS\_NEEDED);  
 sp.setVbarPolicy(ScrollPane.ScrollBarPolicy.AS\_NEEDED);  
 sp.setStyle("-fx-background-color: transparent; -fx-background: #FFFFFF;\n");  
  
 initiateData();  
  
 primaryStage.show();  
 }  
  
 public void drawTask(int procNumber, com.gantt.entity.Task task) {  
  
 Rectangle temp = new Rectangle();  
 temp.setX(START\_X + RATE\_H \* TASK\_WEIGHT \* task.getEntryTime());  
 temp.setY(START\_Y + RATE\_V \* ((procNumber + 1) \* (TASK\_HEIGHT + INTERVAL) - TASK\_HEIGHT \* 2));  
 temp.setWidth(RATE\_H \* TASK\_WEIGHT \* (task.getExitTime() - task.getEntryTime()));  
  
 temp.setHeight(TASK\_HEIGHT \* RATE\_V\*2);  
 temp.setArcWidth(50);  
 temp.setArcHeight(0);  
 temp.setFill(Color.WHITE);  
 temp.setId("bord");  
 temp.setStyle("-fx-stroke: blue; -fx-stroke-width: 2;");  
 root.getChildren().add(temp);  
  
 String capture = Integer.toString(task.getNumber());  
 if (capture.length() < 2)  
 capture = " " + capture;  
 int taskTime= task.getExitTime() - task.getEntryTime();  
 double prob = 0.1;  
 prob = prob \* taskTime/2 + 0.05;  
  
 if(taskTime>1&&taskTime<=3) {  
 prob+=taskTime\*0.05;  
 }  
  
 Text texttemp = new Text(temp.getX() + temp.getWidth() \* prob, temp.getY() + temp.getHeight() \* 0.7, capture);  
  
 texttemp.setFill(Color.BLACK);  
 //texttemp.setRotate(270);  
 texttemp.setFont(Font.font("Monaco", FontWeight.BOLD, 18));  
  
  
 root.getChildren().add(texttemp);  
 }  
  
 public void drawTasks() {  
 for (int i = 0; i < PROC\_NUMBER; i++) {  
 com.gantt.entity.Processor p = model.getProcessor(i);  
 for (int j = 0; j < p.size(); j++) {  
 drawTask(i, p.getTask(j));  
 }  
 }  
 }  
  
 public void drawAxes() {  
 for (int i = 0; i < PROC\_NUMBER; i++) {  
 com.gantt.entity.Processor p = model.getProcessor(i);  
 for (int j = 0; j < p.size(); j++) {  
 if (p.getTask(j).getExitTime() > MAX\_TIME)  
 MAX\_TIME = p.getTask(j).getExitTime();  
 }  
 }  
  
 Line line10 = new Line();  
 line10.setStartX(START\_X);  
 line10.setStartY(START\_Y);  
 line10.setEndX(START\_X);  
 line10.setEndY(START\_Y);  
 line10.setStyle("-fx-stroke: green; ");  
 root.getChildren().add(line10);  
  
  
 END\_X = START\_X + RATE\_H \* TASK\_WEIGHT \* MAX\_TIME \* 1.05;  
 //double c =RATE\_H\*TASK\_WEIGHT\*(1-END\_X%(RATE\_H \* TASK\_WEIGHT\*1.05));  
 END\_Y = START\_Y + RATE\_V \* PROC\_NUMBER \* (TASK\_HEIGHT + INTERVAL) \* 1.05;  
  
 for (int i = 0; i < PROC\_NUMBER; i++) {  
 Line lineTemp = new Line();  
 lineTemp.setStartX(START\_X);  
 lineTemp.setStartY(START\_Y + RATE\_V \* (i + 1) \* (TASK\_HEIGHT + INTERVAL));  
 lineTemp.setEndX(END\_X);  
 lineTemp.setEndY(START\_Y + RATE\_V \* (i + 1) \* (TASK\_HEIGHT + INTERVAL));  
 lineTemp.setStyle("-fx-stroke: green; ");  
 root.getChildren().add(lineTemp);  
 }  
  
 int istart = 1;  
 int iinc = 1;  
 int idesc = 5;  
  
 for (int i = istart; i <= MAX\_TIME; i=i+iinc) {  
 Line lineTemp = new Line();  
 lineTemp.setStartX(START\_X + RATE\_H \* TASK\_WEIGHT \* i);  
 lineTemp.setStartY(START\_Y-3);  
 lineTemp.setEndX(START\_X + RATE\_H \* TASK\_WEIGHT \* i);  
 lineTemp.setEndY(END\_Y);  
 lineTemp.setStyle("-fx-stroke: green;");  
  
 if (i%idesc==0) {  
 String capture = Integer.toString(i);  
 if (capture.length() < 2)  
 capture = " " + capture;  
 Text texttemp = new Text(START\_X + RATE\_H \* TASK\_WEIGHT \* (i-0.15)-7,START\_Y-8, capture);  
 texttemp.setFill(Color.rgb(25, 25, 25));  
 texttemp.setFont(Font.font("Monaco", FONT\_HEIGHT));  
 //lineTemp.setStyle("-fx-stroke: #8c8c8c;");  
  
 root.getChildren().add(texttemp);  
  
 }  
 root.getChildren().add(lineTemp);  
 }  
  
 // Y axis  
 Line line1 = new Line();  
 line1.setStartX(START\_X);  
 line1.setStartY(START\_Y);  
 line1.setEndX(END\_X-3);  
 line1.setEndY(START\_Y);  
 line1.setStrokeWidth(1);  
 line1.setStyle("-fx-stroke: red; -fx-stroke-width: 1;");  
 root.getChildren().add(line1);  
  
 // X axis  
 Line line2 = new Line();  
 line2.setStartX(START\_X);  
 line2.setStartY(START\_Y);  
 line2.setEndX(START\_X);  
 line2.setEndY(END\_Y-3);  
 line2.setStrokeWidth(1);  
 line2.setStyle("-fx-stroke: red; -fx-stroke-width: 1;");  
 root.getChildren().add(line2);  
  
  
 // Right arrow  
 double[] xint1 = {-15, -15};  
 double[] yint1 = {-5, 5};  
 for (int i = 0; i < xint1.length; i++) {  
 Line l1 = new Line();  
 l1.setStartX(END\_X);  
 l1.setStartY(START\_Y);  
 l1.setEndX(END\_X + xint1[i]);  
 l1.setEndY(START\_Y + yint1[i]);  
 l1.setStrokeWidth(1);  
 l1.setStyle("-fx-stroke: black; -fx-stroke-width: 1;");  
 root.getChildren().add(l1);  
 }  
  
 // Down arrow  
 double[] xint2 = {-5, 5};  
 double[] yint2 = {-15, -15};  
 for (int i = 0; i < xint2.length; i++) {  
 Line l2 = new Line();  
 l2.setStartX(START\_X);  
 l2.setStartY(END\_Y);  
 l2.setEndX(START\_X + xint2[i]);  
 l2.setEndY(END\_Y + yint2[i]);  
 l2.setStrokeWidth(1);  
 l2.setStyle("-fx-stroke: black; -fx-stroke-width: 1;");  
 root.getChildren().add(l2);  
 }  
  
 for (int i = 0; i < PROC\_NUMBER; i++) {  
 String capture = "Processor "+Integer.toString(i);  
 if (capture.length() < 2)  
 capture = " " + capture;  
 Text texttemp = new Text(0.1 \* START\_X, START\_Y + RATE\_V \* ((i + 1) \* (TASK\_HEIGHT + INTERVAL) - 0.3 \* TASK\_HEIGHT), capture);  
 texttemp.setFill(Color.rgb(0, 0, 0));  
 texttemp.setFont(Font.font("Monaco", FONT\_HEIGHT-5));  
  
  
 root.getChildren().add(texttemp);  
 }  
  
 Text texttemp = new Text(START\_X + (END\_X-START\_X)/2.5+5,START\_Y-50, "Time line");  
 texttemp.setFill(Color.rgb(0, 0, 0));  
 texttemp.setFont(Font.font("Monaco", FONT\_HEIGHT+5));  
  
  
 root.getChildren().add(texttemp);  
 }  
}

package com;  
  
import java.util.\*;  
  
  
public class APN\_MH {  
 private Graph topologyGraph;  
 private List<Link> links;  
 private List<Processor> processors;  
 private int P;  
 private LinkedList<Processor>[][] routes;  
  
 public APN\_MH(Graph topologyGraph) {  
 this.topologyGraph = topologyGraph;  
 this.links = topologyGraph.getLinks();  
 this.processors = topologyGraph.getProcessors();  
 P = processors.size();  
 Routing routing = new Routing(topologyGraph);  
 routes = new LinkedList[P][P];  
  
 for (int i = 0; i < P; i++) {  
 routing.execute(processors.get(i));  
 for (int j = 0; j < P; j++) {  
 routes[i][j] = routing.getPath(processors.get(j));  
 }  
 }  
 }  
  
 public int[][] schedule(ArrayList<Task> tasks) {  
 TreeSet<Task> readyTasks = new TreeSet<>(new Comparator<Task>() {  
 @Override  
 public int compare(Task o1, Task o2) {  
 int priority = o1.getSL() - o2.getSL();  
 if (priority == 0) {  
 priority = o1.getChildren().size() - o2.getChildren().size();  
 if (priority == 0) {  
 double r = Math.random();  
 if (r < 0.5) {  
 priority = -1;  
 }  
 else {  
 priority = 1;  
 }  
 }  
 }  
 return priority;  
 }  
 });  
  
 for (Task taskTask : tasks) {  
 taskTask.setSL(getSL(taskTask));  
  
 if (taskTask.getParents().size() == 0) {  
 readyTasks.add(taskTask);  
 }  
 }  
  
 int[] timeStart = new int[tasks.size()];  
 int[] timeEnd = new int[tasks.size()];  
 int[] allocations = new int[tasks.size()];  
 while (readyTasks.size() != 0) {  
 Task taskToAllocate = readyTasks.pollLast();  
 System.out.println(taskToAllocate.getID() + "\t" + taskToAllocate.getSL());  
 ArrayList<Transmission> parents = taskToAllocate.getParents();  
 ArrayList<LinkInterval> busyLinks = new ArrayList<>();  
  
 int[] transmissionPeriod = {Integer.MAX\_VALUE, Integer.MAX\_VALUE};  
 int processorChosen = -1;  
 for (int i = 0; i < P; i++) {  
 ArrayList<LinkInterval> busyLinksForProcessor = new ArrayList<>();  
 int durationForProcessor = taskToAllocate.getWeight();  
 int[] timeForProcessor = processors.get(i).getFreeInterval(0, durationForProcessor);  
 for (Transmission parent : parents) {  
 int parentTaskIndex = tasks.indexOf(parent.getNode());  
 int parentTaskProcessor = allocations[parentTaskIndex];  
 int timeForParent = timeEnd[parentTaskIndex];  
 if (parentTaskProcessor != i) {  
 int transmissionWeight = parent.getWeight();  
 LinkedList<Processor> path = routes[parentTaskProcessor][i];  
 Processor currentProcessor = topologyGraph.getProcessors().get(parentTaskProcessor);  
 for (int k = 1; k < path.size(); k++) {  
 Processor nextProcessor = path.get(k);  
 Link currentLink = links.get(links.indexOf(new Link(" ", currentProcessor, nextProcessor, 1)));  
 int[] intervalForEdge = currentLink.getTimeForLinkTransmission(timeForParent, transmissionWeight);  
 busyLinksForProcessor.add(new LinkInterval(currentLink, intervalForEdge));  
 timeForParent = intervalForEdge[1];  
 currentProcessor = nextProcessor;  
 }  
 }  
 if (timeForParent > timeForProcessor[0]) {  
 timeForProcessor = processors.get(i).getFreeInterval(timeForParent, durationForProcessor);  
 }  
 }  
 if (timeForProcessor[1] < transmissionPeriod[1]) {  
 transmissionPeriod = timeForProcessor;  
 processorChosen = i;  
 busyLinks = busyLinksForProcessor;  
 }  
 }  
 timeStart[tasks.indexOf(taskToAllocate)] = transmissionPeriod[0];  
 timeEnd[tasks.indexOf(taskToAllocate)] = transmissionPeriod[1];  
 allocations[tasks.indexOf(taskToAllocate)] = processorChosen;  
 processors.get(processorChosen).setBusyInterval(transmissionPeriod);  
 for (LinkInterval element: busyLinks) {  
 element.link.setBusyInterval(element.interval);  
 System.out.println(element + "\tзадача " + taskToAllocate.getID());  
 }  
  
 ArrayList<Transmission> children = taskToAllocate.getChildren();  
 for (int j = 0; j < children.size(); j++) {  
 Task child = children.get(j).getNode();  
 if (child.setParentFinished() == 0) {  
 readyTasks.add(child);  
 }  
 }  
 }  
 int[][] result = {allocations, timeStart, timeEnd};  
 return result;  
 }  
  
  
 private int getSL(Task N) {  
 ArrayList<Transmission> children = N.getChildren();  
 int res = 0;  
 for (int i = 0; i < children.size(); i++) {  
 int sl = getSL(children.get(i).getNode());  
 if (sl > res) res = sl;  
 }  
 return res + N.getWeight();  
 }  
  
 private class LinkInterval {  
 Link link;  
 int[] interval;  
  
 public LinkInterval(Link link, int[] interval) {  
 this.link = link;  
 this.interval = interval;  
 }  
  
 @Override  
 public String toString() {  
 return "Link " + link.toString() + "\t[" + (interval[0] + 1) + "-" + interval[1] + "]";  
 }  
 }  
}

package com;  
  
import java.util.Comparator;  
import java.util.Iterator;  
import java.util.TreeSet;  
  
public class Processor {  
 final private String id;  
 final private String name;  
 private int productivity = 1;  
 TreeSet<int[]> busyTime = new TreeSet<>(new Comparator<int[]>() {  
  
 @Override  
 public int compare(int[] o1, int[] o2) {  
 return o1[0] - o2[0];  
 }  
 });  
  
 private static int countOfProcessors = 0;  
  
 public Processor(int productivity) {  
 this("P"+countOfProcessors, "P"+countOfProcessors, productivity);  
 }  
  
 public Processor(String id, String name) {  
 this(id, name, 1);  
 }  
  
 public Processor(String id, String name, int productivity) {  
 this.id = id;  
 this.name = name;  
 this.productivity = productivity;  
  
 countOfProcessors += 1;  
 }  
  
 public String getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getProductivity() {  
 return productivity;  
 }  
  
 public void setProductivity(int productivity) {  
 this.productivity = productivity;  
 }  
  
 @Override  
 public int hashCode() {  
 final int prime = 31;  
 int result = 1;  
 result = prime \* result + ((id == null) ? 0 : id.hashCode());  
 return result;  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj)  
 return true;  
 if (obj == null)  
 return false;  
 if (getClass() != obj.getClass())  
 return false;  
 Processor other = (Processor) obj;  
 if (id == null) {  
 if (other.id != null)  
 return false;  
 } else if (!id.equals(other.id))  
 return false;  
 return true;  
 }  
  
 public int[] getFreeInterval(int start, int duration) {  
  
 int[] interval = {start, start + duration};  
 for (Iterator<int[]> it = busyTime.iterator(); it.hasNext();) {  
 int[] element = it.next();  
 if ((element[0] - interval[0]) >= duration) {  
 break;  
 }  
 else {  
 interval[0] = (interval[0] > element[1]) ? interval[0]:element[1];  
 interval[1] = interval[0] + duration;  
 }  
 }  
 return interval;  
 }  
  
 public void setBusyInterval(int[] interval) {  
 busyTime.add(interval);  
 }  
  
 @Override  
 public String toString() {  
 return name;  
 }  
  
}

package com;  
  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.HashMap;  
import java.util.HashSet;  
import java.util.LinkedList;  
import java.util.List;  
import java.util.Map;  
import java.util.Set;  
  
public class Routing {  
  
 private final List<Processor> nodes;  
 private final List<Link> links;  
 private Set<Processor> settledNodes;  
 private Set<Processor> unSettledNodes;  
 private Map<Processor, Processor> predecessors;  
 private Map<Processor, Integer> distance;  
  
 public Routing(Graph graph) {  
 // create a copy of the array so that we can operate on this array  
 this.nodes = new ArrayList<Processor>(graph.getProcessors());  
 this.links = new ArrayList<Link>(graph.getLinks());  
 }  
  
 public void execute(Processor source) {  
 settledNodes = new HashSet<Processor>();  
 unSettledNodes = new HashSet<Processor>();  
 distance = new HashMap<Processor, Integer>();  
 predecessors = new HashMap<Processor, Processor>();  
 distance.put(source, 0);  
 unSettledNodes.add(source);  
 while (unSettledNodes.size() > 0) {  
 Processor node = getMinimum(unSettledNodes);  
 settledNodes.add(node);  
 unSettledNodes.remove(node);  
 findMinimalDistances(node);  
 }  
 }  
  
 private void findMinimalDistances(Processor node) {  
 List<Processor> adjacentNodes = getNeighbors(node);  
 for (Processor target : adjacentNodes) {  
 if (getShortestDistance(target) > getShortestDistance(node)  
 + getDistance(node, target)) {  
 distance.put(target, getShortestDistance(node)  
 + getDistance(node, target));  
 predecessors.put(target, node);  
 unSettledNodes.add(target);  
 }  
 }  
  
 }  
  
 private int getDistance(Processor node, Processor target) {  
 for (Link link : links) {  
 if (link.getSource().equals(node)  
 && link.getDestination().equals(target)) {  
 return link.getWeight();  
 }  
 }  
 throw new RuntimeException("Should not happen");  
 }  
  
 private List<Processor> getNeighbors(Processor node) {  
 List<Processor> neighbors = new ArrayList<Processor>();  
 for (Link link : links) {  
 if (link.getSource().equals(node)  
 && !isSettled(link.getDestination())) {  
 neighbors.add(link.getDestination());  
 }  
 }  
 return neighbors;  
 }  
  
 private Processor getMinimum(Set<Processor> processors) {  
 Processor minimum = null;  
 for (Processor processor : processors) {  
 if (minimum == null) {  
 minimum = processor;  
 } else {  
 if (getShortestDistance(processor) < getShortestDistance(minimum)) {  
 minimum = processor;  
 }  
 }  
 }  
 return minimum;  
 }  
  
 private boolean isSettled(Processor processor) {  
 return settledNodes.contains(processor);  
 }  
  
 private int getShortestDistance(Processor destination) {  
 Integer d = distance.get(destination);  
 if (d == null) {  
 return Integer.MAX\_VALUE;  
 } else {  
 return d;  
 }  
 }  
  
 /\*  
 \* This method returns the path from the source to the selected target and  
 \* NULL if no path exists  
 \*/  
 public LinkedList<Processor> getPath(Processor target) {  
 LinkedList<Processor> path = new LinkedList<Processor>();  
 Processor step = target;  
 // check if a path exists  
 if (predecessors.get(step) == null) {  
 return null;  
 }  
 path.add(step);  
 while (predecessors.get(step) != null) {  
 step = predecessors.get(step);  
 path.add(step);  
 }  
 if (!path.getLast().equals(target)) {  
 Collections.reverse(path);  
 }  
 return path;  
 }  
  
}

package com;  
  
public class Transmission {  
 private Task task\_;  
 private int weight\_;  
 public Transmission(Task task, int weight){  
 task\_ = task;  
 weight\_=weight;  
  
 }  
 public int getWeight(){  
 return weight\_;  
 }  
 public Task getNode(){  
 return task\_;  
  
  
 }  
}

package com;  
  
import java.util.\*;  
  
public class Link {  
 private final String id;  
 private final Processor source;  
 private final Processor destination;  
 private final int weight;  
 TreeSet<int[]> busyTime = new TreeSet<>(new Comparator<int[]>() {  
  
 @Override  
 public int compare(int[] o1, int[] o2) {  
 return o1[0] - o2[0];  
 }  
 });  
  
 public Link(String id, Processor source, Processor destination, int weight) {  
 this.id = id;  
 this.source = source;  
 this.destination = destination;  
 this.weight = weight;  
 }  
  
 public String getId() {  
 return id;  
 }  
 public Processor getDestination() {  
 return destination;  
 }  
  
 public Processor getSource() {  
 return source;  
 }  
 public int getWeight() {  
 return weight;  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj)  
 return true;  
 if (obj == null)  
 return false;  
 if (getClass() != obj.getClass())  
 return false;  
 Link other = (Link) obj;  
 if (!source.equals(other.getSource()) || !destination.equals(other.getDestination()))  
 return false;  
 return true;  
 }  
  
 @Override  
 public String toString() {  
 return source + "-" + destination;  
 }  
  
 public int[] getTimeForLinkTransmission(int startTime, int duration) {  
 int[] interval = {startTime, startTime + duration};  
 NavigableSet<int[]> times = new TreeSet<>();  
 try {  
 times = busyTime.headSet(busyTime.higher(interval), false);  
 }  
 catch (Exception e) {  
 times = busyTime;  
 }  
  
 for (Iterator<int[]> it = times.descendingIterator(); it.hasNext();) {  
 int[] element = it.next();  
 if (element[0] <= startTime && element[1] > startTime) {  
 interval[0] = element[1];  
 interval[1] = interval[0] + duration;  
 }  
 }  
 return interval;  
 }  
  
 public void setBusyInterval(int[] interval) {  
 busyTime.add(interval);  
 }  
}