Розрахунково-графічна робота

Алгоритм визначення найкоротшого шляху

для пасажира метрополітену

Виконав студент АІ-226
Якубовський М.Р.

Мета роботи :

Розробити та реалізувати алгоритм для знаходження найкоротшого маршруту в метрополітені між заданими станціями та оцінити його ефективність.

Визначити переваги та недоліки всіх алгоритмів.

Опис вхідних та вихідних даних:

Вхідні дані

-граф підземних комунікацій(вершинами графа виступають станції, а ребрами шляхи між станціями);

-початкова станція;

-кінцева станція;

-за основу взяті реальні карти метро Лондона(з кількістю вершин 280 та кількістю ребер 310) та карти метро Києва (з кількістю вершин 53 та кількістю ребер 52);

Вихідні дані

-найкоротший шлях (назви станцій);

-довжина або час найкоротшого шляху;

-час виконання алгоритму;

Теоретичні відомості

- Алгоритм Дейкстри
- Алгоритм Флойда-Уоршелла

Алгоритм Дейкстри

- Алгоритм Дейкстри є одним із ключових методів для знаходження найкоротших шляхів у зважених невід’ємних графах. Цей алгоритм широко використовується в телекомунікаціях, маршрутизації мереж, а також у багатьох інших областях. Основна ідея полягає у поступовому визначенні найкоротших відстаней від вихідного вузла до всіх інших вузлів графа.

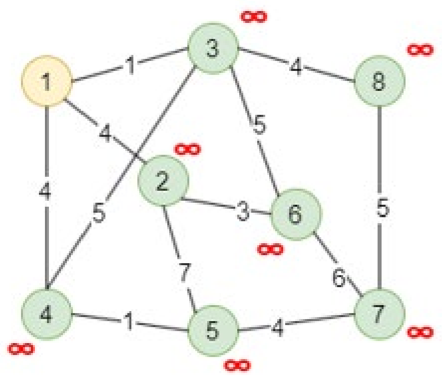
Алгоритм Дейкстри використовує жадібний підхід

Обираючи на кожному кроці найкоротший шлях до вузла, що ще не включений до множини оптимальних шляхів

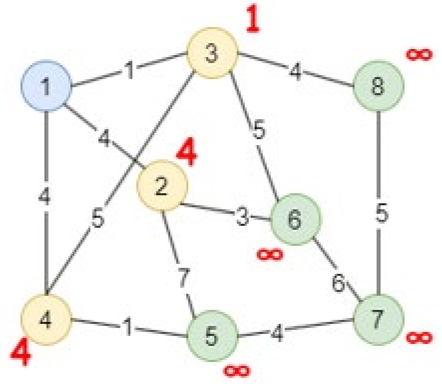
Для кожного вузла ведеться відстеження поточної найкоротшої відстані та оновлюється в разі знаходження коротшого шляху через інший вузол

Алгоритм завершує свою роботу, коли всі вузли графа були розглянуті, і оптимальні шляхи до всіх вузлів визначені.

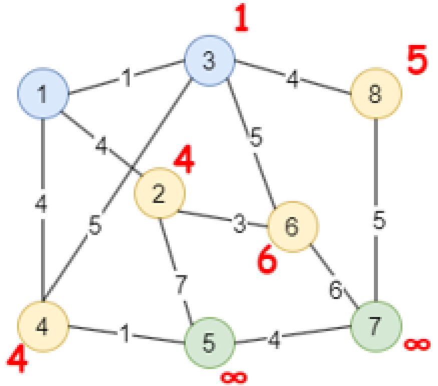
Розглянемо на практиці роботу алгоритму



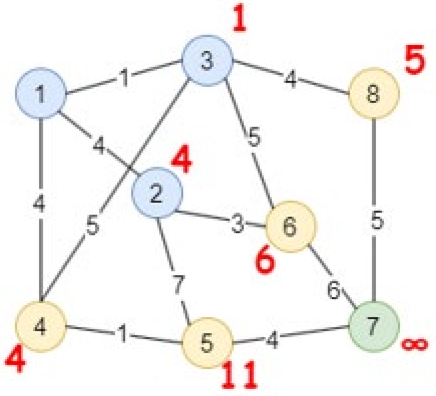
Початкова вершина - 1



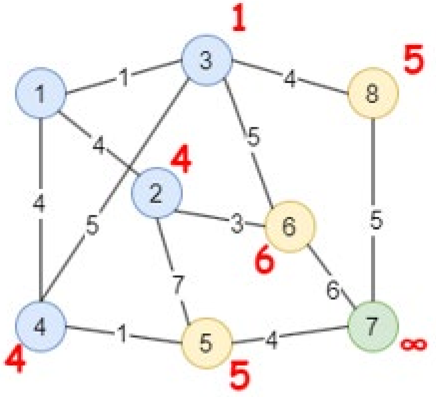
З першої вершини маємо шляхи до 2, 3, 4



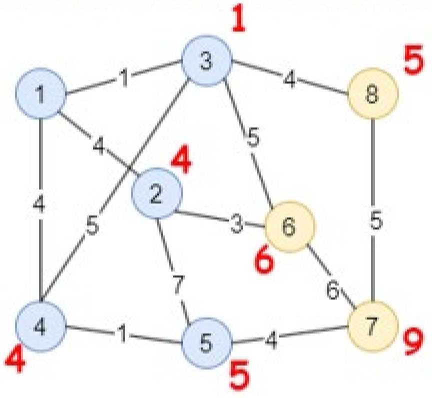
Найменша вага у вершини 3, тому перевіряємо шляхи з неї. Додаємо шлях до 6, 8 вершин



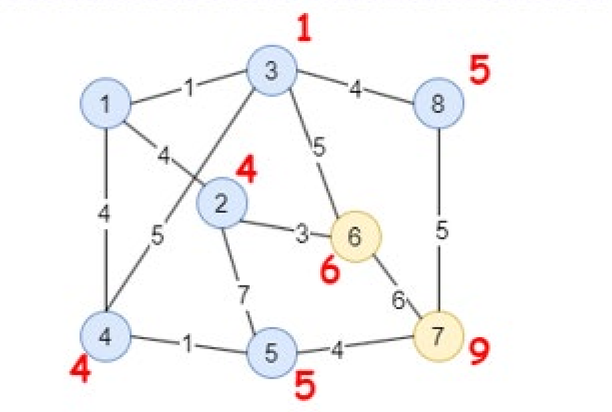
Далі перевіряємо 2 вершину, додаємо шлях до 5



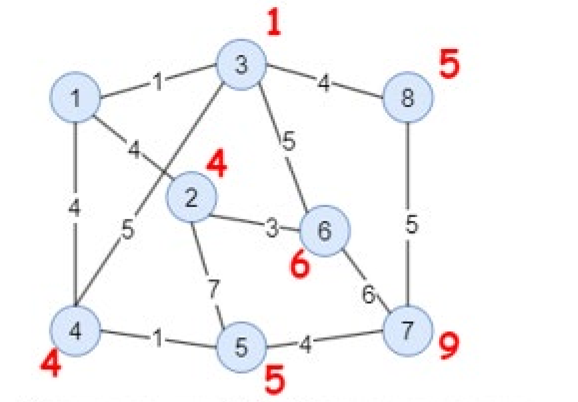
Перевіряємо наступною 4 вершину. Через неї можна скоротити шлях до 5



Перевіряємо 5 вершину, додаємо шлях до 7



Далі перевіряємо 8 вершину. Менших шляхів немає



Остання вершина 6. З неї теж немає коротших шляхів. Отже, граф пройдено

Часова складність:

Алгоритм Дейкстри має часову складність O(|V|^2), де |V| – кількість вершин у графі.

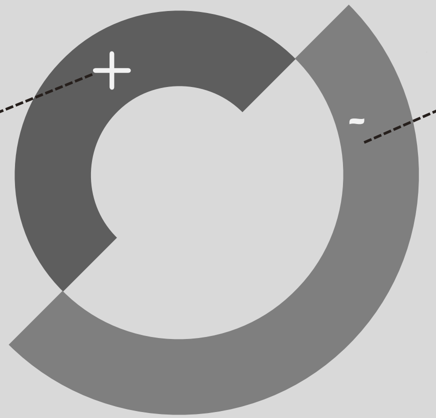
Це робить його ефективним для малих і середніх розмірів графів.

Переваги та недоліки:

Недоліки

не працює на графах з вагами ребер, які можуть бути від'ємними

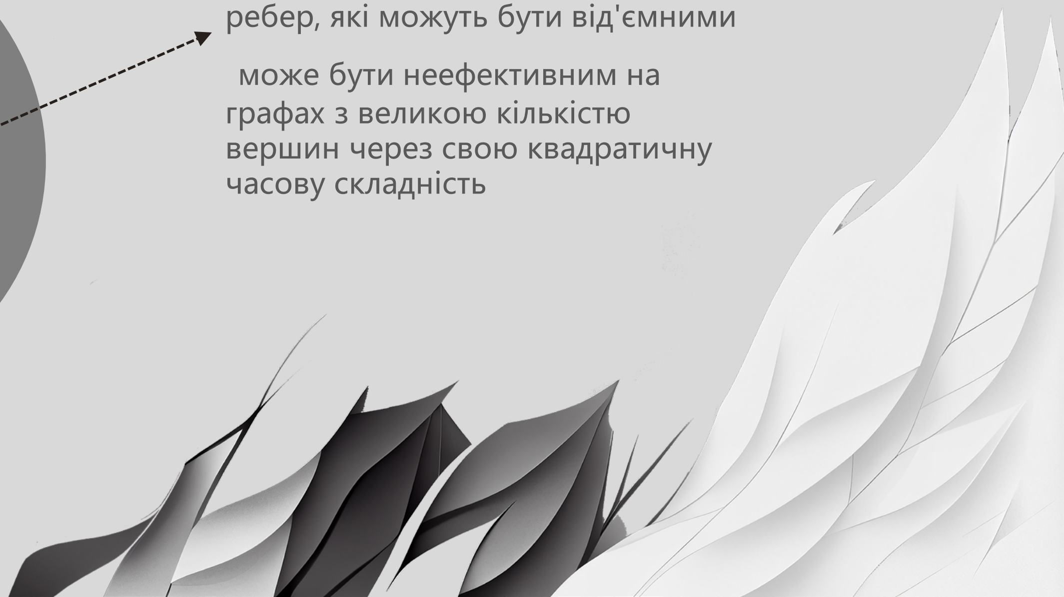
може бути неефективним на графах з великою кількістю вершин через свою квадратичну часову складність



Переваги

надзвичайно ефективний на графах з невеликою кількістю вершин

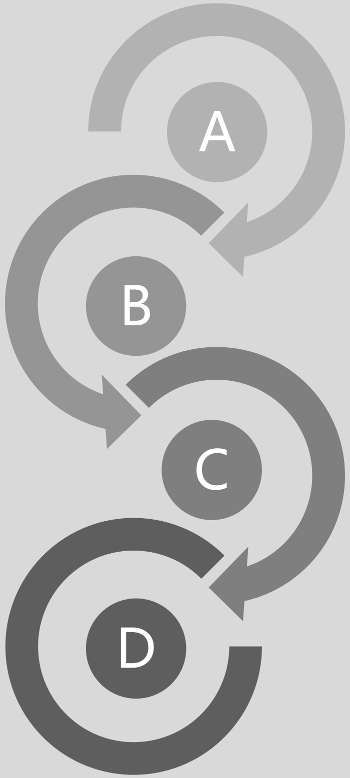
гарантує знаходження найкоротшого шляху в графі з невід'ємними вагами



Алгоритм Флойда

- Алгоритм Флойда, також відомий як алгоритм Варшалла, є алгоритмом для знаходження найкоротших шляхів між всіма парами вершин у орієнтованому графі. Однією з основних його характеристик є універсальність, оскільки він працює як для графів з вагами, так і для графів без ваг. Алгоритм використовує динамічне програмування та ітераційний підхід для ефективного вирішення задачі знаходження найкоротших шляхів.

Основна ідея алгоритму

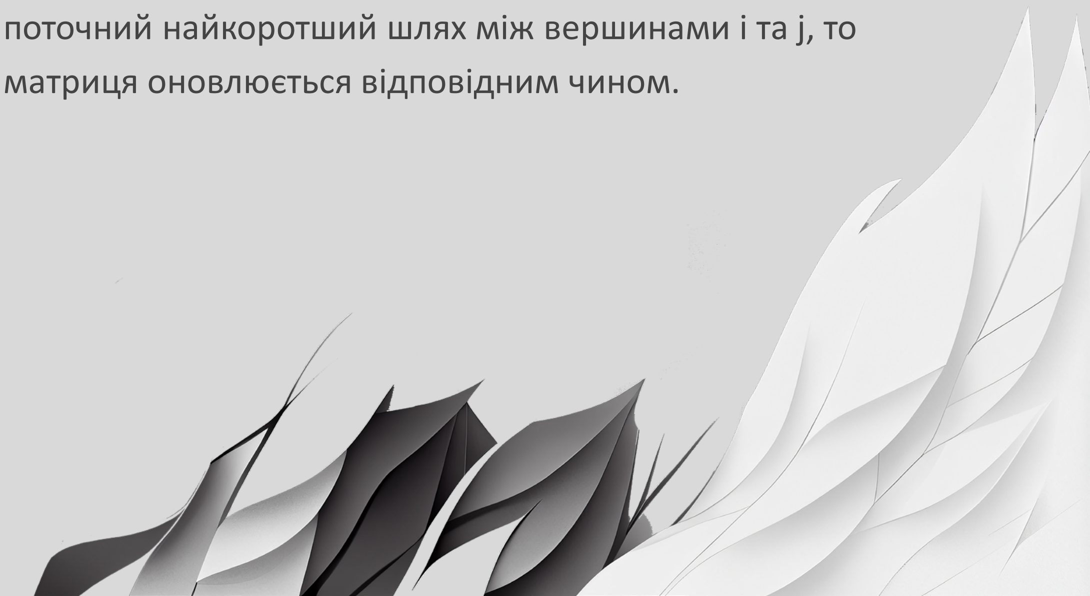


Полягає у тому, щоб побудувати матрицю, в якій кожен елемент (i, j) представляє найкоротший шлях від вершини i до вершини j

Алгоритм поступово покращує цю матрицю, використовуючи проміжні вершини. На кожному кроці алгоритм розглядає можливі шляхи через вершини від 1 до k, де k - це ітерація алгоритму.

Якщо шлях через вершину k виявляється коротшим, ніж поточний найкоротший шлях між вершинами i та j, то матриця оновлюється відповідним чином.

Повторення цього процесу для всіх вершин дозволяє знаходити найкоротші шляхи між усіма парами вершин у графі.



Розглянемо на практиці роботу алгоритму

	X1	X2	X3	X4	X5	X6			1	2	3	4	5	6
X1	-	4	∞	1	∞	7		1	-	2	3	4	5	6
X2	∞	-	2	∞	3	∞		2	1	-	3	4	5	6
X3	∞	2	-	5	∞	∞		3	1	2	-	4	5	6
X4	∞	∞	5	-	∞	∞		4	1	2	3	-	5	6
X5	∞	3	∞	∞	-	2		5	1	2	3	4	-	6
X6	7	∞	∞	6	2	-		6	1	2	3	4	5	-

	X1	X2	X3	X4	X5	X6			1	2	3	4	5	6
X1	-	4	∞	1	∞	7		1	-	2	3	4	5	6
X2	∞	-	2	∞	3	∞		2	1	-	3	4	5	6
X3	∞	2	-	5	∞	∞		3	1	2	-	4	5	6
X4	∞	∞	5	-	∞	∞		4	1	2	3	-	5	6
X5	∞	3	∞	∞	-	2		5	1	2	3	4	-	6
X6	7	11	∞	6	2	-		6	1	1	3	4	5	-

	X1	X2	X3	X4	X5	X6			1	2	3	4	5	6
X1	-	4	6	1	7	7		1	-	2	2	4	2	6
X2	∞	-	2	∞	3	∞		2	1	-	3	4	5	6
X3	∞	2	-	5	5	∞		3	1	2	-	4	2	6
X4	∞	∞	5	-	∞	∞		4	1	2	3	-	5	6
X5	∞	3	5	∞	-	2		5	1	2	2	4	-	6
X6	7	11	13	6	2	-		6	1	1	2	4	5	-

	X1	X2	X3	X4	X5	X6			1	2	3	4	5	6
X1	-	4	6	1	7	7		1	-	2	2	4	2	6
X2	∞	-	2	7	3	∞		2	1	-	3	3	5	6
X3	∞	2	-	5	5	∞		3	1	2	-	4	2	6
X4	∞	7	5	-	10	∞		4	1	3	3	-	3	6
X5	∞	3	5	10	-	2		5	1	2	2	3	-	6
X6	7	11	13	6	2	-		6	1	1	2	4	5	-

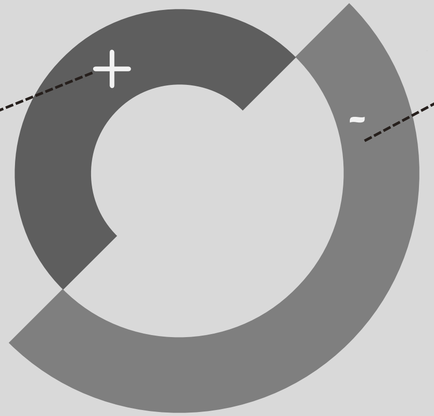
	X1	X2	X3	X4	X5	X6			1	2	3	4	5	6
X1	-	4	6	1	7	7		1	-	2	2	4	2	6
X2	∞	-	2	7	3	5		2	1	-	3	3	5	5
X3	∞	2	-	5	5	7		3	1	2	-	4	2	5
X4	∞	7	5	-	10	12		4	1	3	3	-	3	5
X5	∞	3	5	10	-	2		5	1	2	2	3	-	6
X6	7	5	7	6	2	-		6	1	5	5	4	5	-

	X1	X2	X3	X4	X5	X6			1	2	3	4	5	6
X1	-	4	6	1	7	7		1	-	2	2	4	2	6
X2	12	-	2	7	3	5		2	6	-	3	3	5	5
X3	14	2	-	5	5	7		3	6	2	-	4	2	5
X4	19	7	5	-	10	12		4	6	3	3	-	3	5
X5	9	3	5	8	-	2		5	6	2	2	6	-	6
X6	7	5	7	6	2	-		6	1	5	5	4	5	-

Часова складність:

Часова складність алгоритму Флойда залежить від кількості вершин у графі. Якщо граф має N вершин, то часова складність алгоритму Флойда становить O(N^3). Це означає, що час, необхідний для виконання алгоритму, зростає пропорційно кубу кількості вершин у графі.

Переваги та недоліки:



Переваги

алгоритм працює для різних типів графів.

Цей алгоритм досить простий у реалізації та зрозумілий. Його можна легко використовувати для вирішення задач знаходження найкоротших шляхів у графах.

Недоліки

Часова складність алгоритму Флойда становить O(N^3), де N - кількість вершин у графі. Для великих графів це може бути неефективно.

Алгоритм вимагає матриці розміром N × N для зберігання найкоротших відстаней між парами вершин. Це призводить до великого обсягу використовуваної пам'яті, що особливо важливо для великих графів.

Програмна реалізація алгоритмів

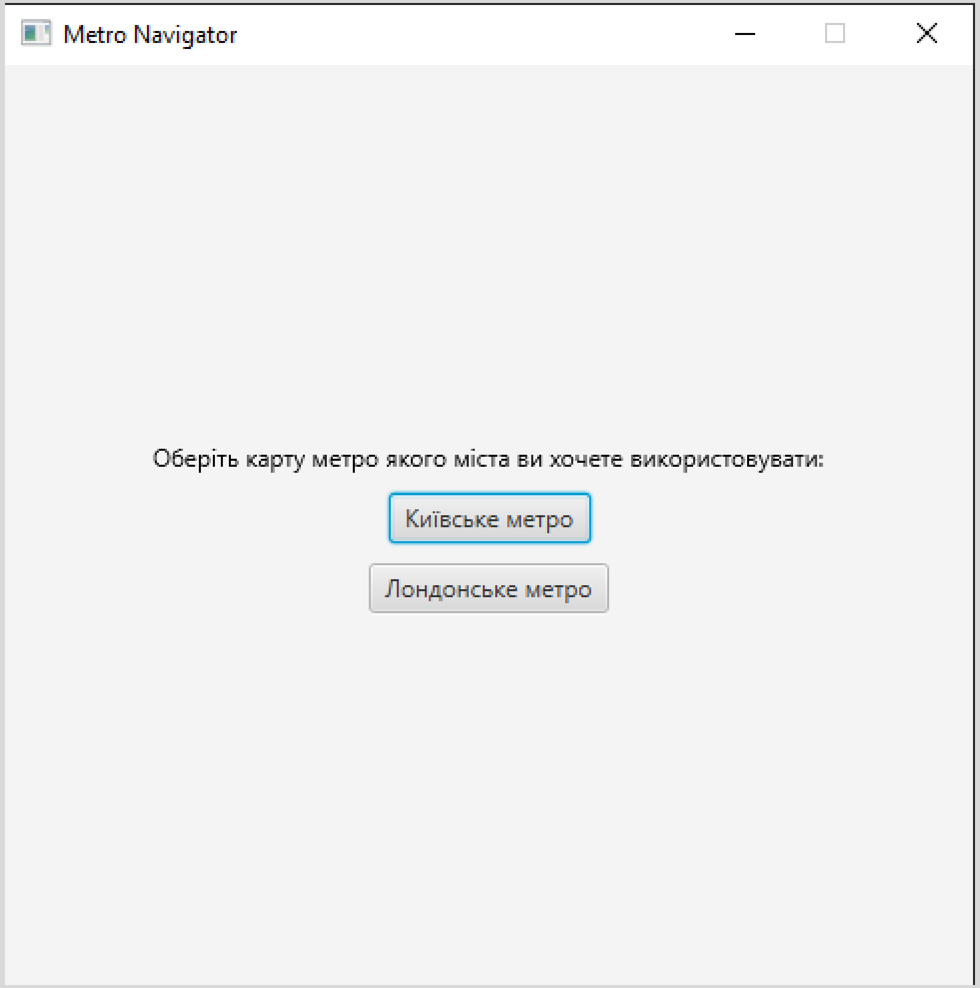
Візуалізація вхідних та вихідних даних

Програмну реалізацію алгоритмів реалізовано на мові Java. Алгоритми написані у вигляді окремих класів для зручності використання.

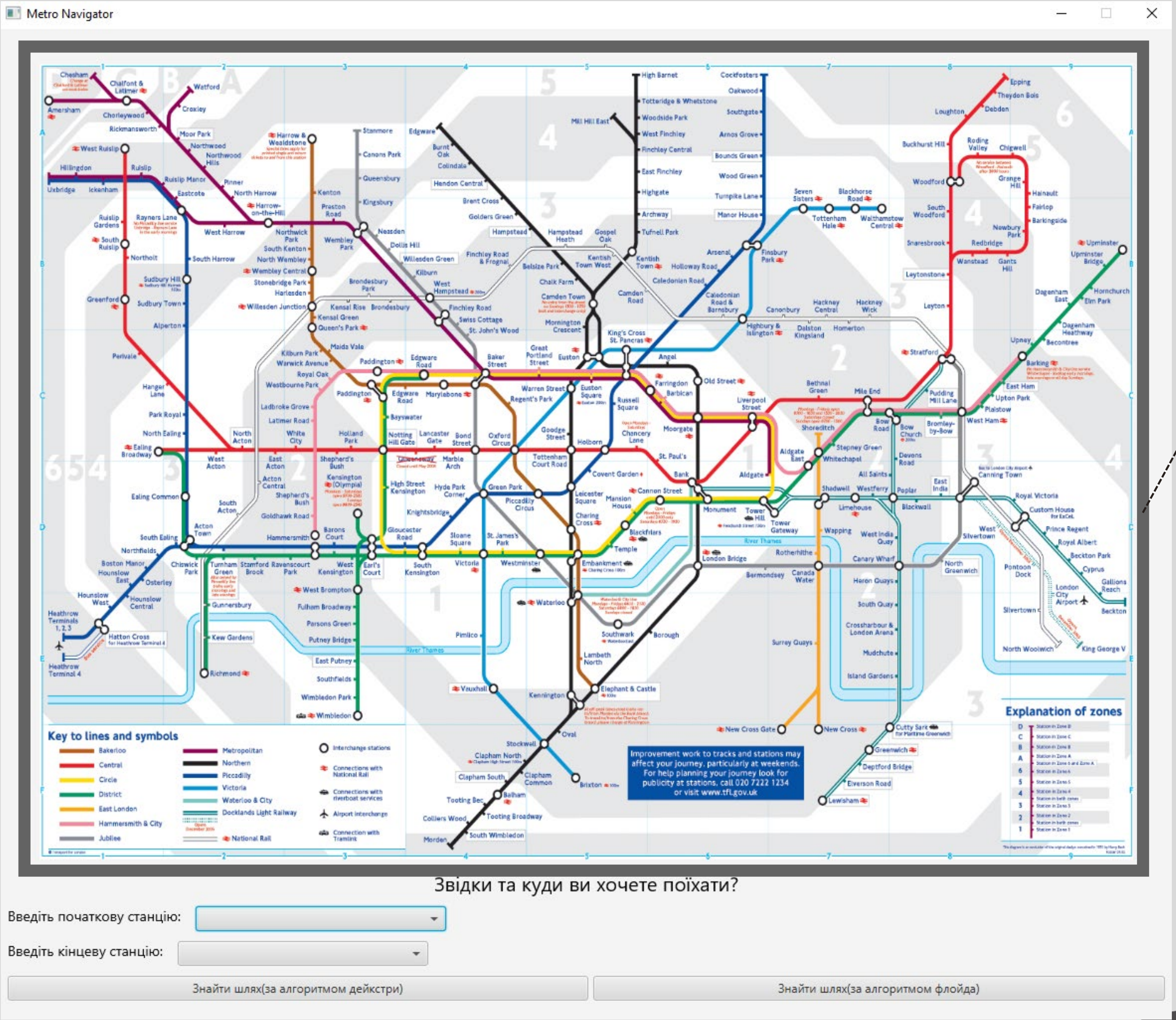
Для реалізації самого додатку навігації на карті метро використанно бібліотеку Java FX.

Розглянемо роботу програми та код алгоритмів.

При запуску програми додаток запитує яку карту
метро використовувати.



При натискані на кнопку визивається нова сцена з відповідною картою метро та ініціалізується
(зараз для прикладу оберемо метро Лондона)



Нова сцена представляє можливість побачити карту самого метро, рисунок визивається з папки ресурсів додатку та є лише імітацією , візуальним прикладом

Сам граф реалізований у вигляді списку та всі вершини і ребра додавалися вручну(на цій карті приблизно 280 вершин та 310 ребер).

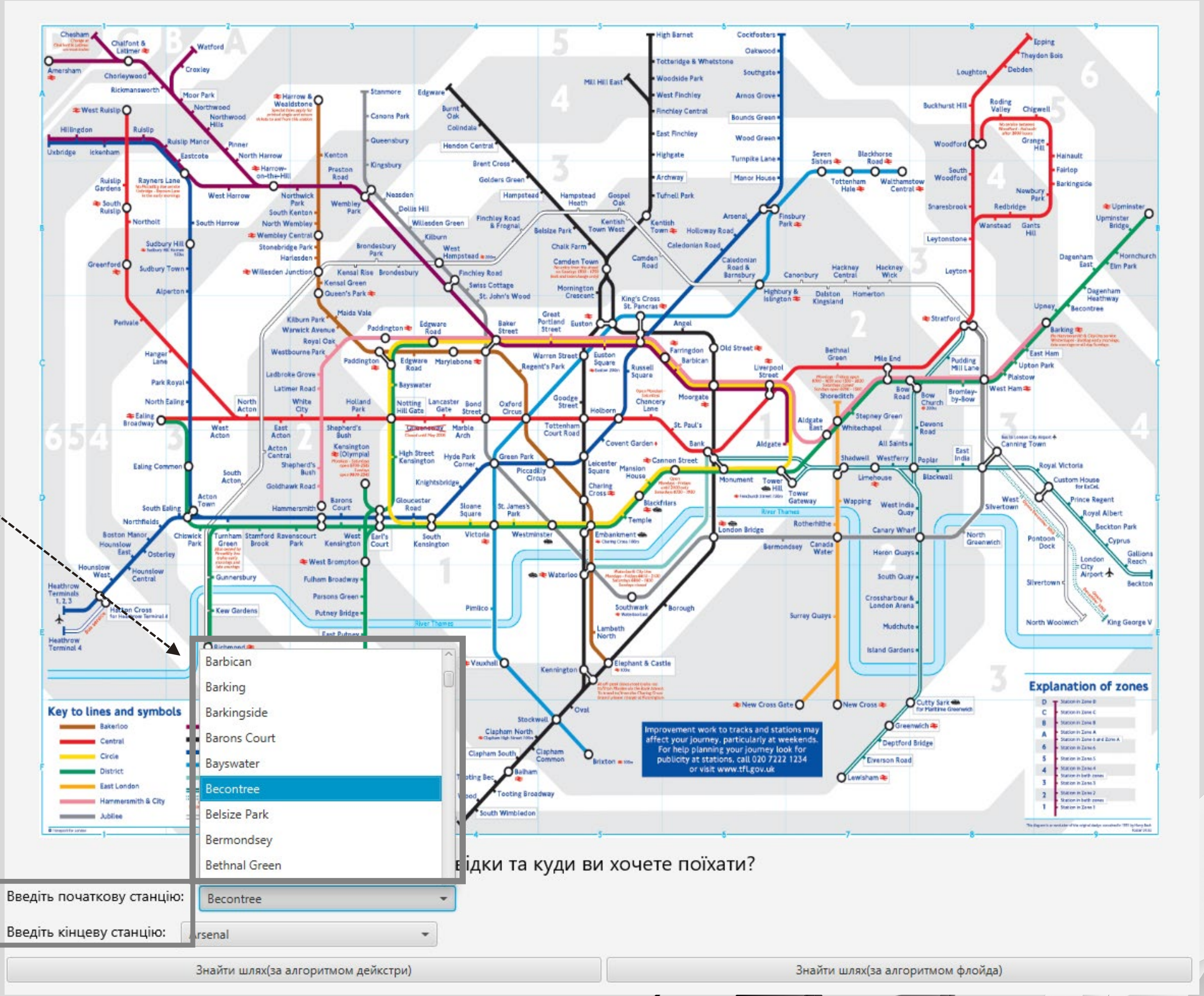
```
11 usages
private final Map<String, Map<String, Integer>> stationConnections;
```

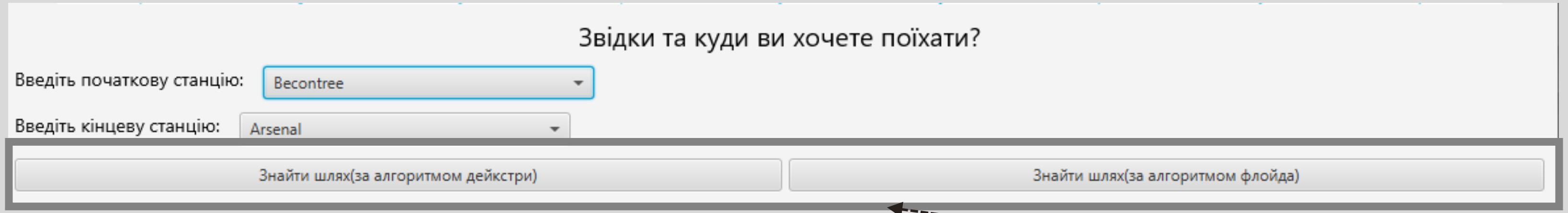
```
// Метод для додавання нової станції до метро
325 usages  Maxyakubovskiy
public void addStation(String stationName) {
    stationConnections.put(stationName, new HashMap<>());
}

// Метод для додавання з'єднання між двома станціями та визначення їхньої відстані
361 usages  Maxyakubovskiy
public void addConnection(String station1, String station2, int distance) {
    stationConnections.get(station1).put(station2, distance);
    stationConnections.get(station2).put(station1, distance);
}
```

Тепер користувачу надається на вибір список всіх станцій

Користувач має вибрати початкову та кінцеву станцію





Далі користувач має вибрати за яким алгоритмом має працювати програма
на вибір дві кнопки з відповідними алгоритмами

Розрахунок на запит користувача виконується при натисканні на кнопку кожного разу

Розглянемо обидва алгоритми окремо

При натискані на кнопку з алгортмом Дейкстри:

1. Викликається метод знаходження найкоротшої дистанції та знаходиться час виконання коду. В метод передається потрібна карта метро та вибрані стартова та кінцеві станції.

2. Викликається метод знаходження найкоротшого шляху(метод відновлює маршрут, використовуючи відомі найкоротші відстані. Заповнюється поле з точками маршруту, починаючи від кінцевої станції і переходячи до початкової.Після цього список перевертається, щоб мати правильний порядок від початкової до кінцевої станції).Потім список додається в спеціальний клас , який виведе шлях у вигляді списку з прокруткою.

3. Вся інформація додається до сцени та в консоль виводиться час виконання алгоритму

4. Дані виводяться на екран користувача

Код алгоритму Дейкстри:

Метод знаходження найкоротшої дистанції

```java
// Метод для знаходження найкоротшого шляху між двома станціями
1 usage  Maxyakubovskiy *
public int findShortestDistance(Metro metro, String startStation, String endStation) {
    Map<String, Integer> shortestDistances = new HashMap<>();
    Set<String> unvisitedStations = new HashSet<>(metro.getStations());// невідвідані станції
    // Ініціалізація всіх станцій з нескінченною довжиною шляху
    for (String station : metro.getStations()) {
        shortestDistances.put(station, Integer.MAX_VALUE);
    }
    // Встановлення початкової станції з довжиною шляху 0
    shortestDistances.put(startStation, 0);
    while (!unvisitedStations.isEmpty()) {
        // Вибір станції з найменшою відомою довжиною шляху
        String currentStation = null;
        for (String station : unvisitedStations) {
            if (currentStation == null || shortestDistances.get(station) < shortestDistances.get(currentStation)) {
                currentStation = station;
            }
        }
        if (currentStation == null || shortestDistances.get(currentStation) == Integer.MAX_VALUE) {
            break;
        }
        // Видалення поточної станції зі списку невідвіданих
        unvisitedStations.remove(currentStation);
        // Оновлення відомих довжин шляху до сусідніх станцій
        for (String neighbor : metro.getConnectedStations(currentStation)) {
            int distanceToNeighbor = metro.getDistanceBetweenStations(currentStation, neighbor);
            int totalDistance = shortestDistances.get(currentStation) + distanceToNeighbor;

            // Оновлення довжини шляху, якщо новий шлях є коротшим
            if (totalDistance < shortestDistances.get(neighbor)) {
                shortestDistances.put(neighbor, totalDistance);
            }
        }
    }
    // Отримання найкоротшого шляху та його довжини
    findShortestPath(metro, startStation, endStation, shortestDistances);
    return shortestDistance;
}
```

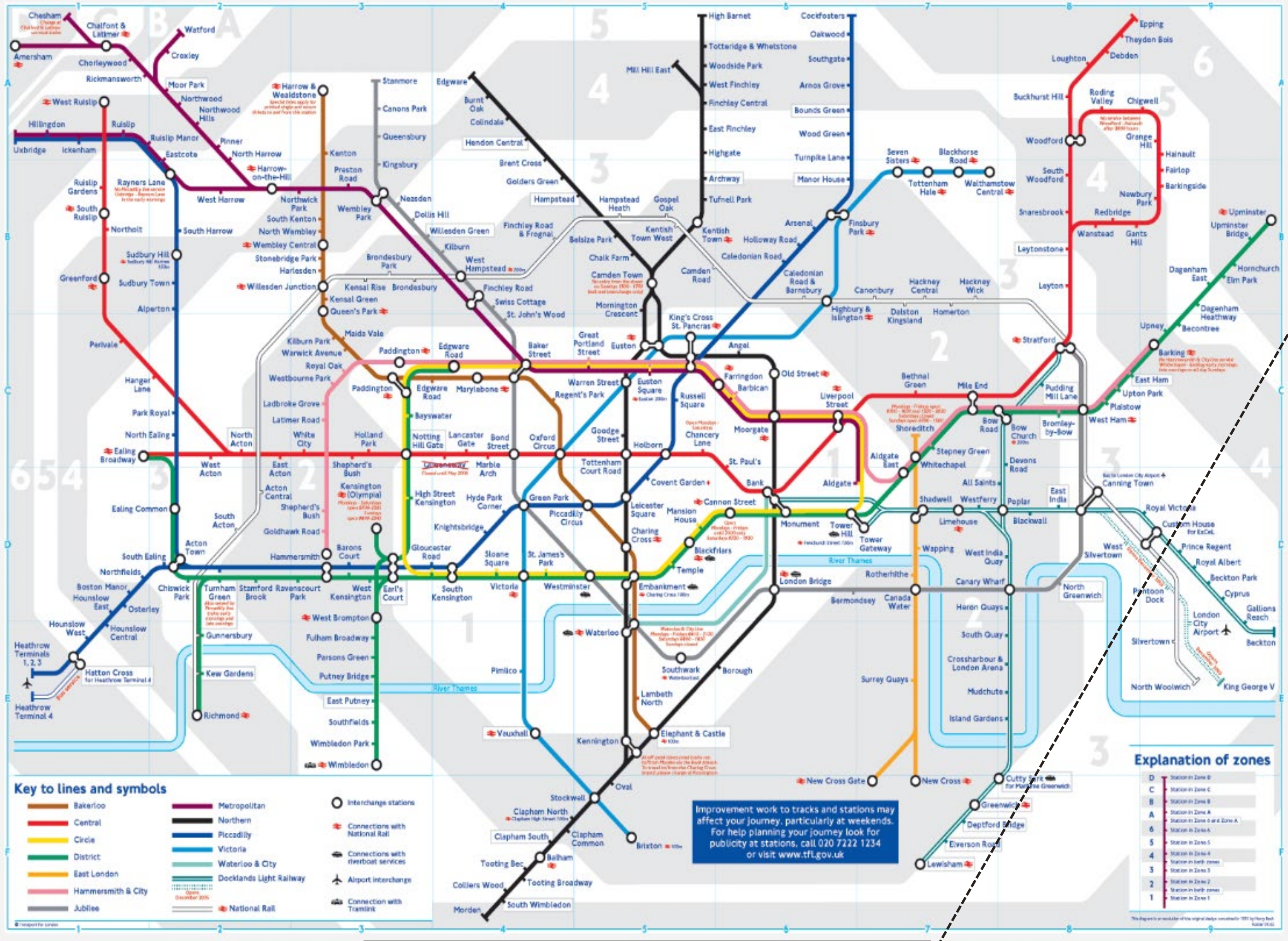
```java
// Метод для знаходження точок найкоротшого шляху та його довжини
1 usage  Maxyakubovskiy *
public void findShortestPath(Metro metro, String startStation, String endStation, Map<String, Integer> shortestDistances) {
    shortestPath.clear();
    shortestDistance = shortestDistances.get(endStation);

    // Відновлення найкоротшого шляху, починаючи з кінцевої станції
    String currentStation = endStation;
    while (!currentStation.equals(startStation)) {
        shortestPath.add(currentStation);
        for (String neighbor : metro.getConnectedStations(currentStation)) {
            int distanceToNeighbor = metro.getDistanceBetweenStations(currentStation, neighbor);
            int totalDistance = shortestDistances.get(currentStation) - distanceToNeighbor;

            // Перехід до попередньої станції в шляху
            if (totalDistance == shortestDistances.get(neighbor)) {
                currentStation = neighbor;
                break;
            }
        }
    }
    shortestPath.add(startStation);
    // Перевернення списку, щоб мати правильний порядок від початкової до кінцевої станції
    Collections.reverse(shortestPath);
}
// Метод для отримання списку точок найкоротшого шляху
1 usage  Maxyakubovskiy
public List<String> getShortestPath() {
    return shortestPath;
}
```

Метод знаходження найкоротшого шляху

Metro Navigator



Найкоротша відстань з Becontree до Arsenal: 58

По такому шляху :

Becontree
Upney
Barking
East Ham
Upton Park

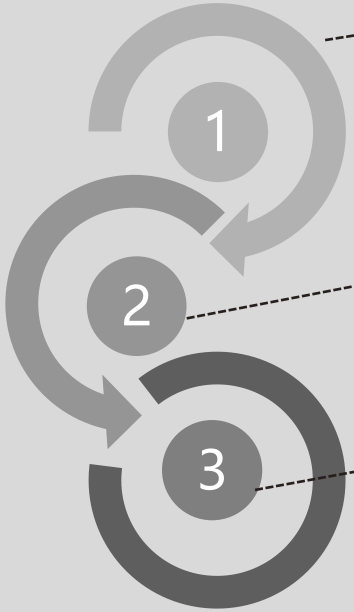
Вивід найкоротшої дистанції між станціями

Вивід найкоротшого шляху між станціями

Вивід часу роботи програми

```
Time Taken: 7672900 nanoseconds
```

При натискані на кнопку з алгортмом Флойда:

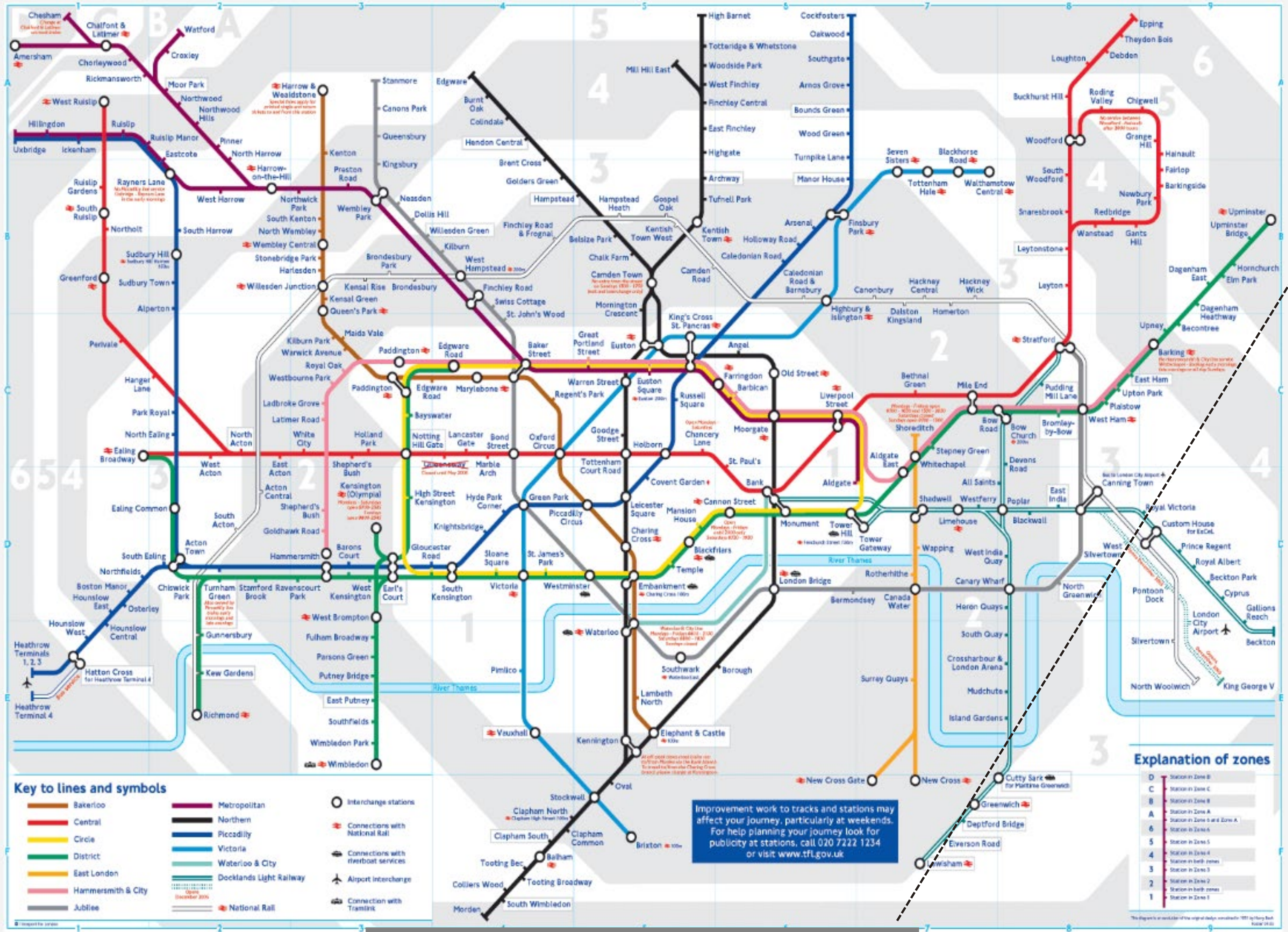


1. Викликається метод знаходження найкоротшої дистанції та знаходиться час виконання коду. В метод передається потрібна карта метро та вибрані стартова та кінцеві.
2. Вся інформація додається до сцени та в консоль виводиться час виконання алгоритму
3. Дані виводяться на екран користувача

Код алгоритму Флойда:

```java
// Метод для знаходження найкоротшої відстані між двома станціями за допомогою алгоритму Флойда-Уоршалла
1 usage  Maxyakubovskiy *
public int findShortestDistance(Metro metro, String startStation, String endStation) {
    int numStations = metro.getStations().size();
    // Матриця для зберігання довжин найкоротших шляхів між станціями
    int[][] distances = new int[numStations][numStations];
    // Ініціалізація початкових відстаней
    for (int i = 0; i < numStations; i++) {
        for (int j = 0; j < numStations; j++) {
            // Встановлення відстані між станціями на нескінченність, крім випадку, коли це одна і та ж станція
            if (i == j) {
                distances[i][j] = 0;
            } else {
                distances[i][j] = Integer.MAX_VALUE;
            }
        }
    }
    // Заповнення початкових відстаней на основі інформації з метро
    for (String station : metro.getStations()) {
        int stationIndex = metro.getStations().indexOf(station);
        for (String neighbor : metro.getConnectedStations(station)) {
            int neighborIndex = metro.getStations().indexOf(neighbor);
            int distance = metro.getDistanceBetweenStations(station, neighbor);
            distances[stationIndex][neighborIndex] = distance;
            distances[neighborIndex][stationIndex] = distance;
        }
    }
```

```java
    // Виконання алгоритму Флойда-Уоршалла для знаходження найкоротших шляхів
    for (int k = 0; k < numStations; k++) {
        for (int i = 0; i < numStations; i++) {
            for (int j = 0; j < numStations; j++) {
                if (distances[i][k] != Integer.MAX_VALUE && distances[k][j] != Integer.MAX_VALUE
                        && distances[i][k] + distances[k][j] < distances[i][j]) {
                    distances[i][j] = distances[i][k] + distances[k][j];
                }
            }
        }
    }
    // Отримання індексів початкової та кінцевої станцій
    int startIndex = metro.getStations().indexOf(startStation);
    int endIndex = metro.getStations().indexOf(endStation);
    // Повернення найкоротшої відстані між початковою та кінцевою станціями
    return distances[startIndex][endIndex];
}
```



Найкоротша відстань з Arnos Grove до Epping: 85

Вивід найкоротшої дистанції між станціями

Вивід часу роботи програми

Time Taken: 33617200 nanoseconds

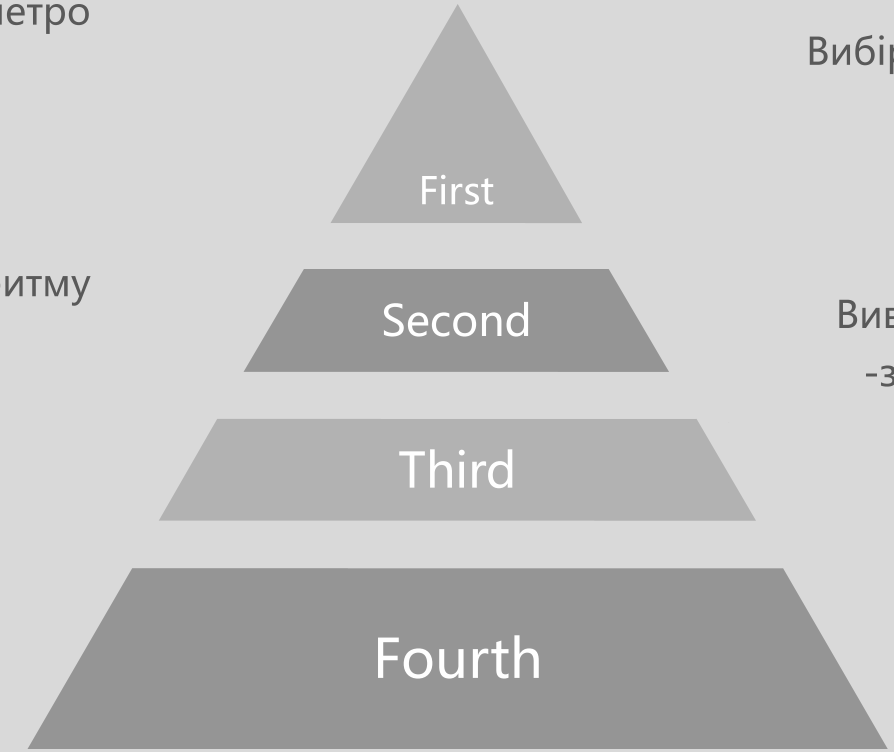
Отже програма має під собою

1 Вибір карти метро

2 Вибір початкової та кінцевої станції

3 Вибір алгоритму

4 Вивід на екран результатів:
-за алгоритмом Дейкстри дистанції та шляху
-за алгоритмом Флойда дистанції



Аналіз отриманих результатів та висновки щодо досягнення мети РГР

Виходячи з тестування двох алгоритмів маємо таку інформацію

В середньому алгоритм Дейкстри в 4.3 рази швидше працює ніж алгоритм Флойда. Це викликано тим що алгоритму Флойда приходиться будувати матрицю для всіх станцій (на карті метро Лондона це приблизно 300 на 300), тому алгоритм Флойда виходить не ефективним по часу для знаходження найкоротшого шляху для пасажира метрополітену.

Алгоритм Флойда, хоча відомий своєю надійністю в знаходженні найкоротших шляхів у графах, має обмеження у вигляді квадратичного обсягу пам'яті через зберігання всіх можливих шляхів між усіма парами вершин. Це стає особливою проблемою при опрацюванні великих графів, обмежуючи його застосування в сучасних системах із обмеженим обсягом пам'яті. З іншого боку, алгоритм Дейкстри, завдяки своїй оптимізації, використовує лише лінійний обсяг пам'яті, зберігаючи інформацію лише про поточні найдешевші шляхи до кожної вершини. Це робить його більш ефективним для великих графів і усуває обмеження щодо обсягу пам'яті, що може виникнути при використанні алгоритму Флойда.

Алгоритми Флойда та Дейкстри представляють два різні підходи до визначення найкоротших шляхів у графі, і кожен має свої переваги та недоліки, що залежать від конкретних умов застосування. У випадку визначення найкоротшого шляху для пасажира метрополітену, де граф має велику кількість вершин та ребер, але без від'ємних ваг, важливим є врахування ефективності та можливості виводу пройденого шляху. Алгоритм Дейкстри має квадратичну часову складність O(V^2) та працює ефективно на графах із невеликою та середньою кількістю вершин, в порівнянні з алгоритмом Флойда, який має кубічну часову складність O(V^3).

Крім того, Дейкстрин алгоритм надає можливість виводу пройденого шляху, що є важливим для пасажирів метрополітену, які хочуть знати конкретні станції, які слід відвідати в маршруті. Але, слід зауважити , що алгоритм Флойда надає можливість знаходити оптимальні шляхи для всіх пар вершин після одного виконання, що надає практичну можливість знаходити зразу декілька найкоротших дистанцій між станціями.

Отже, алгоритм Дейкстри підходить краще для пасажира метрополітену

First

За часом виконання алгоритму на практиці та часовою складність

Second

За використанням пам’яті

Third

За змогою вивести шлях