

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ КИЇВСЬКИЙ
ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

КУРСОВА РОБОТА

З дисципліни «Математичне моделювання»

на тему

Перевірка якості роботи громадського транспорту

Виконав:

Студент групи КМ-93

Тетеря Олексій

Перевірів:

Норкін Б.В.

Київ-2023

ПОСТАНОВКА ЗАДАЧІ

Мета роботи - розробка математичного і програмного забезпечення для перевірки роботи громадського транспорту.

Актуальність проблеми - громадським транспортом сьогодні користуються мільйони людей по всьому світу, тому важливо коректно та швидко оцінювати якість його роботи для оптимізації маршрутів, економії часу пасажирів, економії пального тощо.

В ході виконання курсового проекту було визначено наступні критерії якості роботи громадського транспорту, що перевірятиме система:

1. *Максимальна відстань між станціями* - відстань між будь-якими двома станціями не повинна перевищувати граничне число (для даної роботи 800м)
2. *Максимальний час очікування транспорту* - час, протягом якого пасажир очікує на транспортний засіб не має перевищувати граничне число (для даної роботи 5 хв).
3. *Повнота маршрутів* - між будь-якими двома станціями має проходити хоча б один маршрут
4. *Оптимальність маршруту* - маршрут між будь-якими двома станціями має бути найкоротшим або займати часу не більше, ніж найкоротший.

Важливо розуміти, що під час перевірки не враховуються такі фактори як міський трафік, потенційні ДТП, погодні умови і т.д.

В рамках побудованої моделі, дорогою рухається виключно громадський транспорт з фіксованою швидкістю.

ОПИС ОБРАНОГО МАТЕМАТИЧНОГО МЕТОДУ

В даному проєкті, транспортна система представлена зваженим графом виду $G(V, E)$, де V - множина вершин (зупинки транспорту), а E - множина ребер, що їх поєднують (дороги). Граф має множину підграфів $\langle G_1'(V, E), G_2'(V, E), \dots, G_n'(V, E) \rangle$, що уособлює множину маршрутів транспорту. Ваги ребер означають відстань між зупинками.

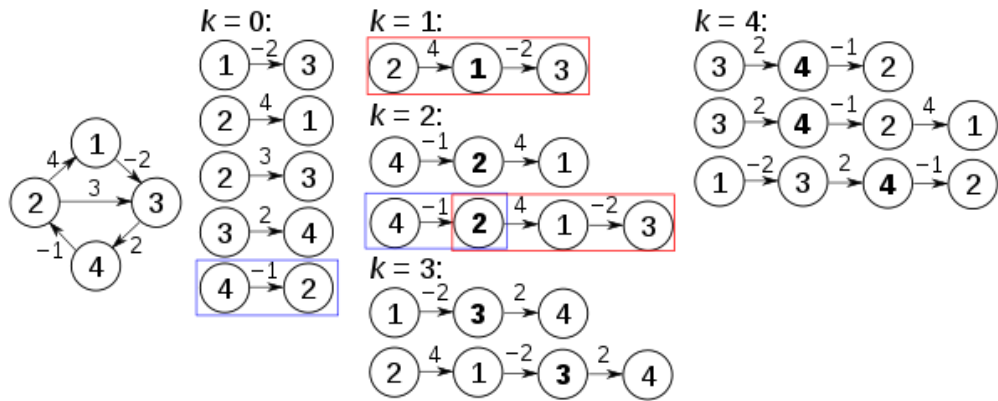
Для реалізації проєкту, були використані наступні математичні методи:

1. **Нормалізація вектора.** Одиничний вектор — це будь-який вектор із довжиною, що дорівнює одиниці; як правило, одиничні вектори використовуються для того, щоб вказувати напрям. Щоб отримати одиничний вектор, вектор довільної довжини можна поділити на його довжину. Цей процес називають нормалізацією вектора. Щоб нормалізувати вектор $\mathbf{a} = (a_1, a_2, a_3)$, його масштабують за допомогою співвідношення з його довжиною $\|\mathbf{a}\|$. Тобто:

$$\hat{\mathbf{a}} = \frac{\mathbf{a}}{\|\mathbf{a}\|} = \frac{a_1}{\|\mathbf{a}\|} \mathbf{e}_1 + \frac{a_2}{\|\mathbf{a}\|} \mathbf{e}_2 + \frac{a_3}{\|\mathbf{a}\|} \mathbf{e}_3$$

2. **Алгоритм Флойда-Воршелла.** Алгоритм Флойда-Воршелла використовується для розв'язання задачі про найкоротший шлях в орієнтованому зваженому графі з додатними або від'ємними вагами ребер (але без негативних циклів). При звичайній реалізації алгоритм повертає довжини (сумарні ваги) найкоротших шляхів між усіма парами вершин. Модифікація алгоритму може також повертати інформацію про самі шляхи. Різні версії алгоритму також

використовуються для знаходження транзитивного замикання у відношенні R , або для знаходження найширшого шляху між усіма парами вершин у зваженому графі.



ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для створення та візуалізації системи була обрана мова програмування Python та її бібліотеки pygame, math, numpy, os. Опис коду представлений нижче:

```
import pygame
import math
import numpy as np
import os
```

Підключення необхідних бібліотек

```
WHITE = (255, 255, 255)
RED = (255, 0, 0)
BLACK = (0, 0, 0)
BLUE = (0, 0, 255)
LIGHT_BLUE = (135, 206, 250)
SPEED = 18
WAITING_MAX = 5
nV = 10
INF = 99
```

Ініціалізація необхідних констант. Кольори, швидкість транспорту, граничний час очікування на транспорт, кількість зупинок та число для методу Флойда-Волшера, яке більше, ніж кожне значення вагів графу

```
pygame.init()
screen = pygame.display.set_mode((900, 900))
```

```
pygame.display.set_caption("Transport system")
```

Ініціалізація та налаштування екрану pygame

```
stop_cords = [[750, 371], [552, 497], [502, 716], [116, 172], [332, 414],  
[392, 176], [284, 612], [335, 782], [712, 620], [193, 488]]  
  
roads_cords = [(4, 9), (0, 5), (0, 1), (0, 8), (4, 3), (5, 4), (9, 5), (4, 1),  
(1, 8), (2, 8), (2, 7), (1, 6), (1, 2), (9, 6), (6, 7)]
```

Ініціалізація списків, що містять координати зупинок громадського транспорту та інформацію про дороги між зупинками

```
roads_distance = []  
  
for item in roads_cords:  
    roads_distance.append(round(math.sqrt((stop_cords[item[1]][0] -  
stop_cords[item[0]][0])**2 + (stop_cords[item[1]][1] -  
stop_cords[item[0]][1])**2) * 2))  
  
roads_time = [round(roads_distance[i]/1000/18*3600/60, 2) for i in  
range(len(roads_distance))]  
  
roads_centers = []  
  
for item in roads_cords:  
    roads_centers.append(((stop_cords[item[1]][0] + stop_cords[item[0]][0])/2,  
(stop_cords[item[1]][1] + stop_cords[item[0]][1])/2))
```

Обчислення відстані між зупинками, часу на подолання дороги між ними з заданою швидкістю та координат середини доріг. Всі значення обчислюються за допомогою відповідних математичних формул. Відстань між зупинками вказана у масштабі 1 піксель = 1 метр

```
for i in range(len(roads_distance)):  
    if roads_distance[i] > 800:  
        print(f'Дорога між зупинками {roads_cords[i]} занадто довга,  
рекомендується додати зупинку')
```

Перевірка критерію №1

```
path = os.path.abspath(__file__)

path = '\\'.join(path.split('\\')[:-1])

bus_img = pygame.image.load(f'{path}/bus.png')
```

Отримання файлу з іконкою громадського транспорту. Рекомендований розмір - від 16x16 до 32x32

```
bus_route1 = [0, 1, 4, 3]

bus_route2 = [5, 0, 8, 2, 7, 6, 9, 5]

bus_route3 = [5, 4, 1, 2, 8]

bus_route4 = [4, 9, 6, 1]

bus_routes = [bus_route1, bus_route2, bus_route3, bus_route4]

bus_quantity = [4, 5, 2, 7]
```

Ініціалізація маршрутів автобусів та списку, що містить кількість автобусів на кожному маршруті

```
all_routes = []

for route in bus_routes:

    rev = list(reversed(route))

    all_routes.append([(route[i-1], route[i]) for i in range(1, len(route))])

    all_routes.append([(rev[i-1], rev[i]) for i in range(1, len(rev))])

all_routes_time = []

for i in range(len(bus_routes)):

    time = 0

    r = [(bus_routes[i][j-1], bus_routes[i][j]) for j in range(1, len(bus_routes[i]))]
```

```

print(f'Маршрут #{i+1}: {r}')

for j in range(len(r)):

    if r[j] in roads_cords:

        time += roads_time[roads_cords.index(r[j])]

    else:

        time += roads_time[roads_cords.index((r[j][1], r[j][0]))]

time = time*2/bus_quantity[i]

all_routes_time.append(round(time, 2))

```

Обчислення часу очікування автобуса на кожному маршруті з урахуванням кількості автобусів.

```

print(f'Максимальний час очікування транспорту')

for i in range(len(all_routes_time)):

    print(f'Маршрут {i+1}: {all_routes_time[i]} хв')

    if all_routes_time[i] > WAITING_MAX:

        print(f'Максимальний час очікування транспорту на маршруті {i+1} більше за заданий поріг. Збільшіть кількість автобусів на маршруті')

```

Перевірка критерію №2 та вивід інформації на екран

```

for item in roads_cords:

    if item not in [a for b in all_routes for a in b]:

        c+=1

        print(f'По цій дорозі не пролягає маршрут: {item}')

if c ==0:

    print('Маршрут пролягає по всім дорогам!')

else:

```



```
print(f'Доріг, по яким не пролягають маршрути: {c}')
```

Перевірка критерію №3 та вивід інформації на екран

```
G = [[0, INF, INF, INF, INF, INF, INF, INF, INF, INF],
      [INF, 0, INF, INF, INF, INF, INF, INF, INF, INF],
      [INF, INF, 0, INF, INF, INF, INF, INF, INF, INF],
      [INF, INF, INF, 0, INF, INF, INF, INF, INF, INF],
      [INF, INF, INF, INF, 0, INF, INF, INF, INF, INF],
      [INF, INF, INF, INF, INF, 0, INF, INF, INF, INF],
      [INF, INF, INF, INF, INF, INF, 0, INF, INF, INF],
      [INF, INF, INF, INF, INF, INF, INF, 0, INF, INF],
      [INF, INF, INF, INF, INF, INF, INF, INF, 0, INF],
      [INF, INF, INF, INF, INF, INF, INF, INF, INF, 0]]

for i in range(len(roads_cords)):
```

```
    G[roads_cords[i][0]][roads_cords[i][1]] =
round(roads_distance[i]/1000/18*3600/60, 2)

    G[roads_cords[i][1]][roads_cords[i][0]] =
round(roads_distance[i]/1000/18*3600/60, 2)
```

Представлення графу транспортної системи у вигляді матриці. Індекс кожного елементу означає номер зупинок, між якими проходить дорога. Значення елементу означає час, що потребує автобус на подолання відстані між зупинками. На головній діагоналі розташовані 0, а час руху між зупинками, між якими немає дороги представлена константою INF.

```
shortest_all = floyd(G)

def floyd(G):

    dist = list(map(lambda p: list(map(lambda q: q, p)), G))

    for r in range(nV):
```

```

    for p in range(nV):

        for q in range(nV):

            dist[p][q] = min(dist[p][q], dist[p][r] + dist[r][q])

    return dist

```

Виклик функції алгоритму Флойда-Волшера, що знаходить найкоротший шлях між кожною парою точок.

```

G1 = [[0, INF, INF, INF, INF, INF, INF, INF, INF, INF],
       [INF, 0, INF, INF, INF, INF, INF, INF, INF, INF],
       [INF, INF, 0, INF, INF, INF, INF, INF, INF, INF],
       [INF, INF, INF, 0, INF, INF, INF, INF, INF, INF],
       [INF, INF, INF, INF, 0, INF, INF, INF, INF, INF],
       [INF, INF, INF, INF, INF, 0, INF, INF, INF, INF],
       [INF, INF, INF, INF, INF, INF, 0, INF, INF, INF],
       [INF, INF, INF, INF, INF, INF, INF, 0, INF, INF],
       [INF, INF, INF, INF, INF, INF, INF, INF, 0, INF],
       [INF, INF, INF, INF, INF, INF, INF, INF, INF, 0]]

```

```

all_routes = [a for b in all_routes for a in b]

```

```

for i in range(len(roads_cords)):

```

```

    if roads_cords[i] in all_routes:

```

```

        G1[roads_cords[i][0]][roads_cords[i][1]] =
G[roads_cords[i][0]][roads_cords[i][1]]

```

```

        G1[roads_cords[i][1]][roads_cords[i][0]] =
G[roads_cords[i][1]][roads_cords[i][0]]

```

```

    elif (roads_cords[i][1], roads_cords[i][0]) in all_routes:

```

```

        G1[roads_cords[i][1]][roads_cords[i][0]] =
G[roads_cords[i][1]][roads_cords[i][0]]

        G1[roads_cords[i][0]][roads_cords[i][1]] =
G[roads_cords[i][0]][roads_cords[i][1]]

shortest_paths = floyd(G1)

```

Представлення графу існуючих автобусних маршрутів у вигляді матриці та виклик функції алгоритму Флойда-Велшера

```

c = 0

add = (nV+1) % 2

for i in range(nV):

    for j in range(i+add,nV):

        if shortest_paths[i][j] > shortest_all[i][j] and i!=j:

            c+=1

            print(f'Шлях з точки {i} в точку {j} неоптимізований! Різниця склала
{round(shortest_paths[i][j] - shortest_all[i][j], 2)} хв.')

if c == 0:

    print('Всі маршрути оптимальні!')

for i in range(nV):

    for j in range(i+add,nV):

        if i !=j:

            print(f'Найкоротший шлях з точки {i} в точку {j} займає
{round(shortest_paths[i][j], 2)} хв.')

```

Порівняння першої та другої матриць Флойда, за допомогою якого перевіряється критерій №4. Різниця в часі між неоптимізованим та найкоротшим маршрутом виводиться на екран. Потім виводиться інформація про найкоротший шлях з кожної точки до кожної.

```

busX = 750

busY = 371

norm = np.array((stop_cords[bus_route1[1]][0]-stop_cords[bus_route1[0]][0],
stop_cords[bus_route1[1]][1] - stop_cords[bus_route1[0]][1]))

norm = norm / np.linalg.norm(norm)

busChangeX = norm[0]/3

busChangeY = norm[1]/3

```

Ініціалізація першого автобусу. Аналогічно для інших трьох маршрутів

```

while run:

    screen.fill(WHITE)

    draw_map()

    busX += busChangeX

    busY += busChangeY

    bus_draw(busX, busY)

```

Візуалізація моделі та зміна положення автобусу за допомогою відповідних функцій

```

def draw_map():

    font = pygame.font.SysFont('comicsans', 40)

    for i in range(len(stop_cords)):

        text = font.render(str(i), True, BLACK)

        screen.blit(text, stop_cords[i])

    font = pygame.font.SysFont('comicsans', 20)

    for i in range(len(roads_distance)):

        text = font.render(str(roads_distance[i]), True, RED)

        screen.blit(text, roads_centers[i])

    for i in range(0, len(roads_cords)):

        pygame.draw.line(screen, BLACK, stop_cords[roads_cords[i][0]],
stop_cords[roads_cords[i][1]], width=4)

```

```

for i in range(0, len(stop_cords)):

    pygame.draw.circle(screen, BLUE, stop_cords[i], 15, width=0)

def bus_draw(x, y):

    screen.blit(bus_img, (x, y))

```

```

for i in range(0, len(bus_route1)):

    if round(busX) == stop_cords[bus_route1[i]][0] and round(busY) ==
stop_cords[bus_route1[i]][1] and round(busX) != stop_cords[bus_route1[-1]][0]
and round(busY) != stop_cords[bus_route1[-1]][1]:

        norm =
np.array((stop_cords[bus_route1[i+1]][0]-stop_cords[bus_route1[i]][0],
stop_cords[bus_route1[i+1]][1]-stop_cords[bus_route1[i]][1]))

        norm = norm / np.linalg.norm(norm)

        busChangeX = norm[0]/3

        busChangeY = norm[1]/3

        if round(busX) == stop_cords[bus_route1[-1]][0] and round(busY) ==
stop_cords[bus_route1[-1]][1]:

            bus_route1 = list(reversed(bus_route1))

            norm =
np.array((stop_cords[bus_route1[i+1]][0]-stop_cords[bus_route1[i]][0],
stop_cords[bus_route1[i+1]][1]-stop_cords[bus_route1[i]][1]))

            norm = norm / np.linalg.norm(norm)

            busChangeX = norm[0]/3

            busChangeY = norm[1]/3


```

Зміна напрямку за допомогою нормалізації вектора при досягненні наступної точки маршруту. Аналогічно для інших маршрутів

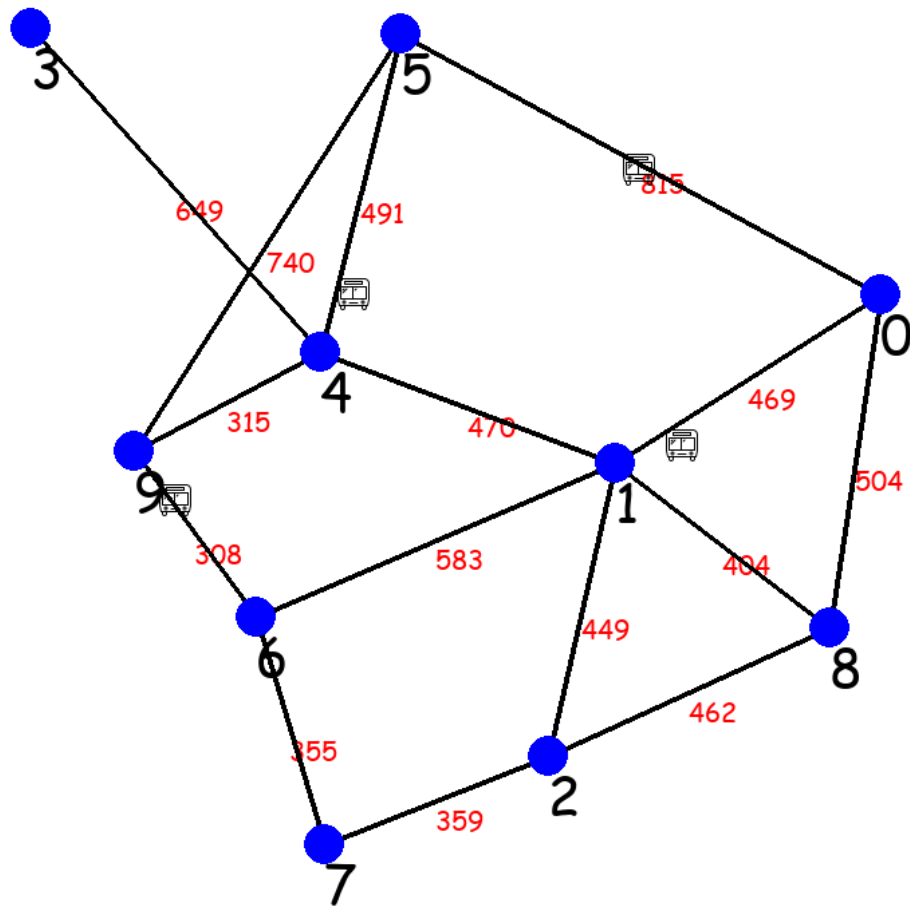
```
for event in pygame.event.get():  
    if event.type == pygame.QUIT:  
        run = False  
  
pygame.display.update()
```

Оновлення екрану для зміни положення автобусів

РЕЗУЛЬТАТ ВИКОНАННЯ ПРОГРАМИ

 Transport system

— □ ×



Візуалізація моделі

Дорога між зупинками (0, 5) занадто довга, рекомендується додати зупинку
Маршрут #1: [(0, 1), (1, 4), (4, 3)]
Маршрут #2: [(5, 0), (0, 8), (8, 2), (2, 7), (7, 6), (6, 9), (9, 5)]
Маршрут #3: [(5, 4), (4, 1), (1, 2), (2, 8)]
Маршрут #4: [(4, 9), (9, 6), (6, 1)]
Максимальний час очікування транспорту
Маршрут 1: 2.65 хв
Маршрут 2: 4.73 хв
Маршрут 3: 6.25 хв
Максимальний час очікування транспорту на маршруті з більше за заданий поріг. Збільшіть кількість автобусів на маршруті
Маршрут 4: 1.15 хв
По цій дорозі не пролягає маршрут: (1, 8)
Доріг, по яким не пролягають маршрути: 1
Шлях з точки 1 в точку 8 неоптимізований! Різниця склала 1.69 хв.
Шлях з точки 3 в точку 8 неоптимізований! Різниця склала 1.69 хв.
Шлях з точки 4 в точку 8 неоптимізований! Різниця склала 1.69 хв.
Шлях з точки 6 в точку 8 неоптимізований! Різниця склала 0.63 хв.
Шлях з точки 8 в точку 9 неоптимізований! Різниця склала 0.98 хв.
Найкоротший шлях з точки 0 в точку 1 займає 1.56 хв.
Найкоротший шлях з точки 0 в точку 2 займає 3.06 хв.
Найкоротший шлях з точки 0 в точку 3 займає 5.29 хв.
Найкоротший шлях з точки 0 в точку 4 займає 3.13 хв.
Найкоротший шлях з точки 0 в точку 5 займає 2.72 хв.
Найкоротший шлях з точки 0 в точку 6 займає 3.5 хв.
Найкоротший шлях з точки 0 в точку 7 займає 4.26 хв.
Найкоротший шлях з точки 0 в точку 8 займає 1.68 хв.
Найкоротший шлях з точки 0 в точку 9 займає 4.18 хв.
Найкоротший шлях з точки 1 в точку 2 займає 1.5 хв.
Найкоротший шлях з точки 1 в точку 3 займає 3.73 хв.
Найкоротший шлях з точки 1 в точку 4 займає 1.57 хв.
Найкоротший шлях з точки 1 в точку 5 займає 3.21 хв.
Найкоротший шлях з точки 1 в точку 6 займає 1.94 хв.
Найкоротший шлях з точки 1 в точку 7 займає 2.7 хв.
Найкоротший шлях з точки 1 в точку 8 займає 3.04 хв.
Найкоротший шлях з точки 1 в точку 9 займає 2.62 хв.

Найкоротший шлях з точки 2 в точку 4	займає 3.07 хв.
Найкоротший шлях з точки 2 в точку 5	займає 4.71 хв.
Найкоротший шлях з точки 2 в точку 6	займає 2.38 хв.
Найкоротший шлях з точки 2 в точку 7	займає 1.2 хв.
Найкоротший шлях з точки 2 в точку 8	займає 1.54 хв.
Найкоротший шлях з точки 2 в точку 9	займає 3.41 хв.
Найкоротший шлях з точки 3 в точку 4	займає 2.16 хв.
Найкоротший шлях з точки 3 в точку 5	займає 3.8 хв.
Найкоротший шлях з точки 3 в точку 6	займає 4.24 хв.
Найкоротший шлях з точки 3 в точку 7	займає 5.42 хв.
Найкоротший шлях з точки 3 в точку 8	займає 6.77 хв.
Найкоротший шлях з точки 3 в точку 9	займає 3.21 хв.
Найкоротший шлях з точки 4 в точку 5	займає 1.64 хв.
Найкоротший шлях з точки 4 в точку 6	займає 2.08 хв.
Найкоротший шлях з точки 4 в точку 7	займає 3.26 хв.
Найкоротший шлях з точки 4 в точку 8	займає 4.61 хв.
Найкоротший шлях з точки 4 в точку 9	займає 1.05 хв.
Найкоротший шлях з точки 5 в точку 6	займає 3.5 хв.
Найкоротший шлях з точки 5 в точку 7	займає 4.68 хв.
Найкоротший шлях з точки 5 в точку 8	займає 4.4 хв.
Найкоротший шлях з точки 5 в точку 9	займає 2.47 хв.
Найкоротший шлях з точки 6 в точку 7	займає 1.18 хв.
Найкоротший шлях з точки 6 в точку 8	займає 3.92 хв.
Найкоротший шлях з точки 6 в точку 9	займає 1.03 хв.
Найкоротший шлях з точки 7 в точку 8	займає 2.74 хв.
Найкоротший шлях з точки 7 в точку 9	займає 2.21 хв.
Найкоротший шлях з точки 8 в точку 9	займає 4.95 хв.

Перевірка критеріїв

ВИСНОВКИ

В ході реалізації курсового проекту було розроблене програмне забезпечення для перевірки якості роботи громадського транспорту. Запропонований варіант реалізації повністю виконує поставлену задачу, працює коректно та швидко.