

Прізвище: Пастернак

Ім'я: Вероніка

Група: КН-406

Варіант: 24

GitHub: <https://github.com/veronikalpnu/8semest-kriviy>

Кафедра: САПР

Дисципліна: Дискретні моделі САПР

Перевірив: Кривий Р.З.



ЗВІТ

до лабораторної роботи №2
на тему: “Алгоритм рішення задачі листоноші”

Мета роботи: вивчити алгоритми рішення задачі листоноші.

Теоретичні відомості:

ЗАДАЧА ЛИСТОНОШІ. ОСНОВНІ ПОНЯТТЯ. ВЛАСТИВОСТІ.

Будь-який листоноша перед тим, як відправитись в дорогу повинен підібрати на пошті листи, що відносяться до його ділянки, потім він повинен рознести їх адресатам, що розмістились вздовж маршрута його проходження, і повернутись на пошту. Кожен листоноша, бажаючи втратити якомога менше сил, хотів би подолати свій маршрут найкоротшим шляхом. Загалом, задача листоноші полягає в тому, щоб пройти всі вулиці маршрута і повернутися в його початкову точку, мінімізуючи при цьому довжину пройденого шляху.

Задача листоноші може бути сформульована в термінах теорії графів. Для цього побудуємо граф $G = (X, E)$, в якому кожна дуга відповідає вулиці в маршруті руху листоноші, а кожна вершина - стик двох вулиць. Ця задача являє собою задачу пошуку найкоротшого маршруту, який включає кожне ребро хоча б один раз і закінчується у початковій вершині руху.

ЕЙЛЕРОВИЙ ЦИКЛ

Ейлеревим циклом в графі називається шлях, який починається і закінчується в тій самій вершині, при чому всі ребра графа проходяться тільки один раз.

Ейлеревим шляхом називається шлях, який починається в вершині A , а закінчується в вершині B , і всі ребра проходяться лише по одному разу.

Граф, який включає в себе ейлерів цикл називається **ейлеревим**.

Теорема 1.

Якщо у графа всі вершини мають парну степінь, то цей граф Ейлерів, тобто має Ейлерів цикл. Також вірне і протилежне твердження.

Якщо в графі існує Ейлерів цикл, то це означає, що в нього всі вершини мають парний степінь.

Теорема 2.

Якщо в графі існує Ейлерів шлях, то це означає, що в нього є строго дві непарні вершини. При чому шлях починається в одній з них, а закінчується в іншій.

КРИТЕРІЙ ІСНУВАННЯ ЕЙЛЕРОВОГО ШЛЯХУ

Якщо G (ейлерів граф, то будь-який його ейлерів цикл неєдиний і відрізняється від інших ейлерових циклів графа G принаймні або зміною початкової вершини і/або зміною порядку проходження.

Для знаходження деякого ейлерового циклу в ейлеровому графі G можна застосувати так званий алгоритм Фльорі. Фіксуємо довільну початкову вершину циклу. На кожному кроці процедури до шуканого циклу обираємо (доки це можливо) те ребро, після вилучення якого граф не розіб'ється на дві нетривіальні зв'язні компоненти. Кожне обране ребро вилучаємо з G .

Процедура завершується, коли всі ребра буде вичерпано. Неважко обґрунтувати, що сформульований алгоритм будує ейлерів цикл графа G .

Виходячи з постановки задачі листоноші:

Очевидно, що в незв'язному графі (в графі, який має декілька компонент) не існує маршруту листоноші (не говорячи про існування ейлерового маршруту).

Очевидно, що кількість приходів листоноші в деяку вершину повинна бути рівною кількості виходів з цієї вершини. При цьому, якщо листоноша не проходить через деяке ребро більше одного разу, то вершина повинна бути інцидентна парній кількості ребер. Загальну кількість ребер інцидентних вершині x , назовемо степінню вершини і будемо позначати через $d(x)$. Якщо в графі G всі вершини мають *парний (непарний)* степінь, то граф називається *парним (непарним)*. В орієнтованому графі число дуг, які входять у вершину x називають *внутрішню степінню* вершини x або *півстепінню входу* (позначається через $d^-(x)$), а які виходять - *зовнішню степінню* вершини x або *півстепінню виходу* (позначається $d^+(x)$). Якщо $d^-(x) = d^+(x)$ то граф називається *симетричним*.

ЗАДАЧА ЛИСТОНОШІ ДЛЯ НЕОРІЄНТОВАНОГО ГРАФА.

Задача (китайського) листоноші — задача пошуку найкоротшого циклу в графі, що включає всі ребра. Існують варіанти задачі для орієнтованих та неорієнтованих графів та для графів, частина ребер яких орієнтована, а частина — ні.

Задача листоноші для неорієнтованого графа $G(X, E)$ - це задача для графа, в якому ребра можна проходити в будь-якому з двох напрямків.

Необхідно розглянути окремо наступні два випадки :

Випадок 1 : Граф G парний.

Випадок 2 : Граф G непарний.

Випадок 1 : Якщо граф парний, то оптимальним рішенням задачі є ейлеровий маршрут. В цьому випадку листоноша не повинен обходити більше одного разу будь-яку вулицю, в даному випадку ребро графа.

Як знайти на графі G єйлеровий маршрут, в якому “ S ” – початкова вершина? Для цього необхідно пройти будь-яке ребро (S, x) інцидентне вершині “ S ”, а потім ще невикористане ребро, інцидентне вершині “ x ”. Кожен раз, коли листоноша приходить в деяку вершину, є невикористане ребро по якому листоноша покидає цю вершину. Дуги, по яким здійснений обхід, створюють цикл $C1$. Якщо в цикл $C1$ ввійшли всі ребра графа G , то $C1$ є єйлеревим маршрутом (оптимальним для задачі).

Випадок 2 : Граф є непарним. В будь-якому маршруті листоноші число входів у вершину рівне числу виходів з неї. Тому, якщо вершина “ x ”, має непарну степінь, то по крайній мірі одне ребро, інцидентне вершині “ x ”, буде обходитись вдруге. Нехай $f(i, j)$ - число додаткових проходжень листоношею ребра (i, j) . Значить ребро (i, j) обходиться $f(i, j) + 1$ раз (число $f(i, j) + 1$ - невід’ємне, ціле число).

Побудуємо новий граф $G^* = (X, E^*)$, в якому ребро (i, j) графа G повторюється $f(i, j) + 1$ раз. Очевидно, що єйлеровий маршрут в графі G^* відповідає маршруту в графі G . Листоноша прагне вибрати значення змінних $f(i, j)$ такими, щоб :

- а) граф G^* був парним;
- б) $\sum (i, j) f(i, j)$ - загальна довжина вдруге пройдених ребер – була мінімальна.

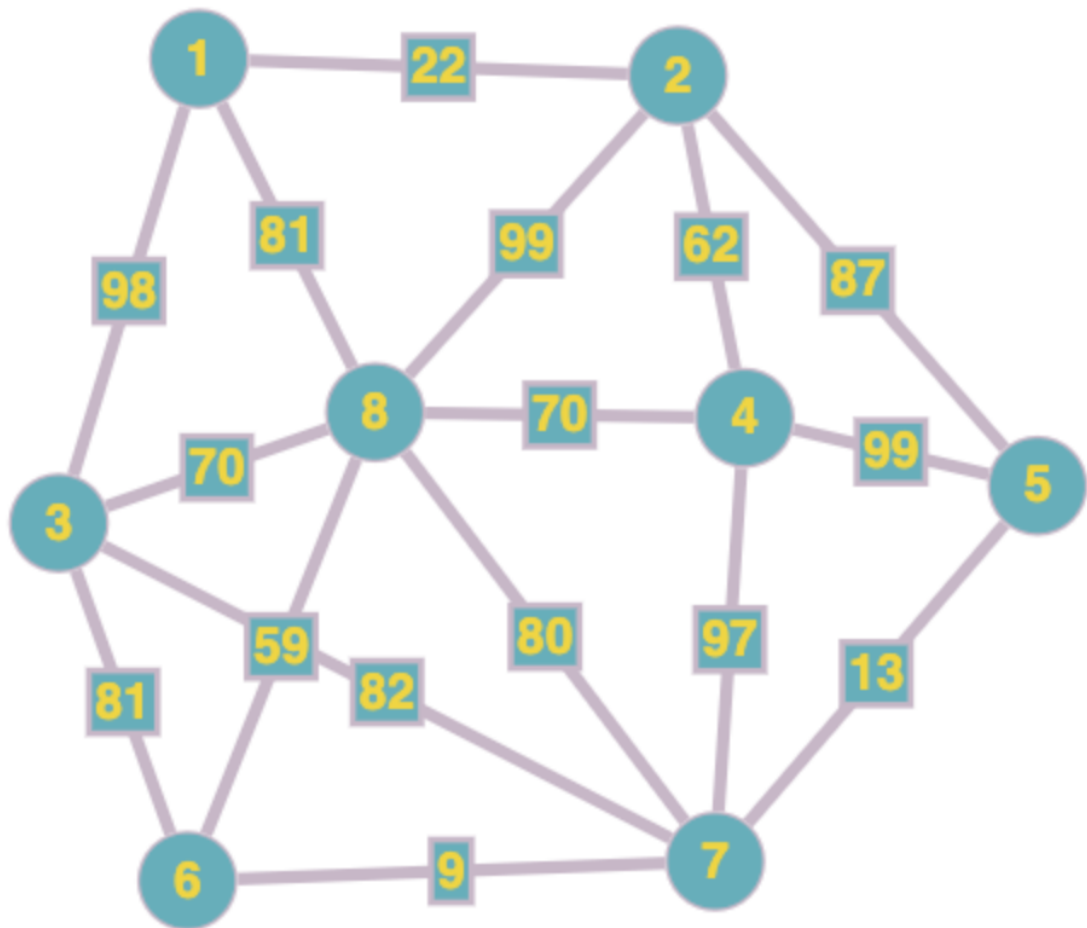
Індивідуальне завдання

- 1) Отримати у викладача індивідуальне завдання.
- 2) Підготувати програму для вирішення виданого завдання.
- 3) Запустити на виконання програму, що розв’язує задачу листоноші.
- 4) Проглянути результат роботи програми. Результат може бути позитивний (шлях знайдено) або негативний (шлях відсутній).
- 5) У випадку, коли шлях знайдено (не знайдено), необхідно модифікувати граф, коректуючи два або три зв’язки таким чином, щоб знайти граф, на якому задача листоноші не розв’язується (розв’язується).
- 6) Здійснити перевірки роботи програм з результатами розрахунків, проведених вручну.
- 7) Зафіксувати результати роботи.
- 8) Оформити і захистити звіт

Ручні обчислення:

Отримані дані згідно індивідуального завдання (l2-3.txt):

	1	2	3	4	5	6	7	8
1	0	22	98	0	0	0	0	81
2	22	0	0	62	87	0	0	99
3	98	0	0	0	0	81	82	70
4	0	62	0	0	99	0	97	70
5	0	87	0	99	0	0	13	0
6	0	0	81	0	0	0	9	59
7	0	0	82	97	13	9	0	80
8	81	99	70	70	0	59	80	0



1) Спочатку визначаємо вершини із непарними степенями:

- Вершина **1** має 3 степінь
- Вершина **5** має 3 степінь
- Вершина **6** має 3 степінь
- Вершина **7** має 5 степінь

2) Формуємо таблицю довжин найкоротших ланцюжків між всіма парами вершин з непарними степенями:

	1	2	3	4	5	6	7	8
1	0	22	98	0	109	131	122	81
2	22	0	0	62	87	0	0	99
3	98	0	0	0	0	81	82	70
4	0	62	0	0	99	0	97	70
5	0	87	0	99	0	22	13	0
6	0	0	81	0	0	0	9	59
7	0	0	82	97	13	9	0	80
8	81	99	70	70	0	59	80	0

- 3) Формуємо паросполучення вершин і рахуємо суму їхніх ваг:
- Вузли $[1; 5]$ та $[6; 7] = 109 + 9 = 118$
 - Вузли $[1; 6]$ та $[5; 7] = 131 + 13 = 144$
 - Вузли $[1; 7]$ та $[5; 6] = 122 + 22 = 144$
- 4) Вибираємо паро сполучення з найменшою сумою - $[1; 5]$ та $[6; 7] = 118$
- 5) Знаходимо ребра, які при обході листиноші будуть повторюватися:
- $[1, 2] = 22$
 - $[2, 5] = 8$
 - $[6, 7] = 9$
- 6) Далі виконуємо обхід листиноші використовуючи ребра для повторного проходження. Для вирішення задачі будемо використовувати два списки:
S – стек відвіданих вершин
C – список вершин занесених в обхід листиноші

Пройдені вершини будемо виключати з графа. Починаємо з вершини 1.

Список ребер графа: $[[1; 2], [1; 3], [1; 8], [2; 4], [2; 5], [2; 8], [3; 6], [3; 7], [3; 8], [4; 5], [4; 7], [4; 8], [5; 7], [6; 7], [6; 8], [7; 8]]$

Крок 1.

$S = [1]$ $C = []$

Крок 2.

Продивляємся ребра, що виходять з вершини 1. Знаходимо перше, яке попалося в списку вершин графа. Вершину поміщаємо в стек S, а ребро видаляємо зі списку ребер.

Ребро, яке видаляється $[1; 2]$

$S = [1, 2]$

$C = []$

Список ребер, що залишилися:

$[[1; 3], [1; 8], [2; 4], [2; 5], [2; 8], [3; 6], [3; 7], [3; 8], [4; 5], [4; 7], [4; 8], [5; 7], [6; 7], [6; 8], [7; 8]]$

Список повторних ребер:

$[[1; 2], [2; 5], [6; 7]]$

Крок 3.

Повторюємо попередній крок.

Ребро – $[2; 4]$

$S = [1, 2, 4]$

$C = []$

Список ребер, що залишилися:

$[[1; 3], [1; 8], [2; 5], [2; 8], [3; 6], [3; 7], [3; 8], [4; 5], [4; 7], [4; 8], [5; 7], [6; 7], [6; 8], [7; 8]]$

Список повторних ребер:

$[[1; 2], [2; 5], [6; 7]]$

Крок 4.

Повторюємо попередній крок.

Ребро – [4; 5]

S = [1, 2, 4, 5]

C = []

Список ребер, що залишилися:

[[1; 3], [1; 8], [2; 5], [2; 8], [3; 6], [3; 7], [3; 8], [4; 7], [4; 8], [5; 7], [6; 7], [6; 8], [7; 8]]

Список повторних ребер:

[[1; 2], [2; 5], [6; 7]]

Крок 5.

Повторюємо попередній крок.

Ребро – [5; 7]

S = [1, 2, 4, 5, 7]

C = []

Список ребер, що залишилися:

[[1; 3], [1; 8], [2; 5], [2; 8], [3; 6], [3; 7], [3; 8], [4; 7], [4; 8], [6; 7], [6; 8], [7; 8]]

Список повторних ребер:

[[1; 2], [2; 5], [6; 7]]

Крок 6.

Повторюємо попередній крок.

Ребро – [7; 8]

S = [1, 2, 4, 5, 7, 8]

C = []

Список ребер, що залишилися:

[[1; 3], [1; 8], [2; 5], [2; 8], [3; 6], [3; 7], [3; 8], [4; 7], [4; 8], [6; 7], [6; 8]]

Список повторних ребер:

[[1; 2], [2; 5], [6; 7]]

Крок 7.

Повторюємо попередній крок.

Ребро – [1; 8]

S = [1, 2, 4, 5, 7, 8, 1]

C = []

Список ребер, що залишилися:

[[1; 3], [2; 5], [2; 8], [3; 6], [3; 7], [3; 8], [4; 7], [4; 8], [6; 7], [6; 8]]

Список повторних ребер:

[[1; 2], [2; 5], [6; 7]]

Крок 8.

Повторюємо попередній крок.

Ребро – [1; 3]

S = [1, 2, 4, 5, 7, 8, 1, 3]

C = []

Список ребер, що залишилися:

[[2; 5], [2; 8], [3; 6], [3; 7], [3; 8], [4; 7], [4; 8], [6; 7], [6; 8]]

Список повторних ребер:

[[1; 2], [2; 5], [6; 7]]

Крок 9.

Повторюємо попередній крок.

Ребро – [3; 6]

S = [1, 2, 4, 5, 7, 8, 1, 3, 6]

C = []

Список ребер, що залишилися:

[[2; 5], [2; 8], [3; 7], [3; 8], [4; 7], [4; 8], [6; 7], [6; 8]]

Список повторних ребер:

[[1; 2], [2; 5], [6; 7]]

Крок 10.

Повторюємо попередній крок.

Ребро – [6; 7]

S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7]

C = []

Список ребер, що залишилися:

[[2; 5], [2; 8], [3; 7], [3; 8], [4; 7], [4; 8], [6; 8]]

Список повторних ребер:

[[1; 2], [2; 5], [6; 7]]

Крок 11.

Повторюємо попередній крок.

Ребро – [3; 7]

S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3]

C = []

Список ребер, що залишилися:

[[2; 5], [2; 8], [3; 8], [4; 7], [4; 8], [6; 8]]

Список повторних ребер:

[[1; 2], [2; 5], [6; 7]]

Крок 12.

Повторюємо попередній крок.

Ребро – [3; 8]

S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8]

C = []

Список ребер, що залишилися:

[[2; 5], [2; 8], [4; 7], [4; 8], [6; 8]]

Список повторних ребер:

[[1; 2], [2; 5], [6; 7]]

Крок 13.

Повторюємо попередній крок.

Ребро – [2; 8]

S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8, 2]

C = []

Список ребер, що залишилися:

[[2; 5], [4; 7], [4; 8], [6; 8]]

Список повторних ребер:

[[1; 2], [2; 5], [6; 7]]

Крок 14.

Повторюємо попередній крок.

Ребро – [2; 5]

S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8, 2, 5]

C = []

Список ребер, що залишилися:

[[4; 7], [4; 8], [6; 8]]

Список повторних ребер:

[[1; 2], [2; 5], [6; 7]]

Крок 15.

Повторюємо попередній крок, але на цьому кроці потрібно вже пройти по повторному ребру.

Ребро – [2; 5]

S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8, 2, 5, 2]

C = []

Список ребер, що залишилися:

[[4; 7], [4; 8], [6; 8]]

Список повторних ребер:

[[1; 2], [6; 7]]

Крок 16.

Повторюємо попередні кроки, на цьому кроці також потрібно вже пройти по повторному ребру.

Ребро – [1; 2]

S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8, 2, 5, 2, 1]

C = []

Список ребер, що залишилися:

[[4; 7], [4; 8], [6; 8]]

Список повторних ребер:

[[6; 7]]

Крок 17.

На цьому кроці можна побачити, що виходу з цієї вершини вже немає у списках ребер, що залишилися, тому ми перекидаємо останню вершину зі списка S в список C доти, доки не з'явиться вершина, з якої буде вихід.

Список ребер, що залишилися:

[[4; 7], [4; 8], [6; 8]]

Список повторних ребер:

[[6; 7]]

Крок 18.

S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8, 2, 5, 2]

C = [1]

Крок 19.

S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8, 2, 5]

C = [1, 2]

Крок 20.

S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8, 2]

C = [1, 2, 5]

Крок 21.

$S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8]$

$C = [1, 2, 5, 2]$

Крок 22.

З вершини 8 вже є ребро, яке від неї відходить.

Ребро – $[4; 8]$

$S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8, 4]$

$C = [1, 2, 5, 2]$

Список ребер, що залишилися:

$[[4; 7], [6; 8]]$

Список повторних ребер:

$[[6; 7]]$

Крок 23.

З вершини 4 теж є ребро, яке від неї відходить.

Ребро – $[4; 7]$

$S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8, 4, 7]$

$C = [1, 2, 5, 2]$

Список ребер, що залишилися:

$[[6; 8]]$

Список повторних ребер:

$[[6; 7]]$

Крок 24.

З вершини 7 теж є ребро, яке від неї відходить. Це ребро вже проходиться повторно.

Ребро – $[6; 7]$

$S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8, 4, 7, 6]$

$C = [1, 2, 5, 2]$

Список ребер, що залишилися:

$[[6; 8]]$

Список повторних ребер:

$[]$

Крок 25.

З вершини 6 теж є ребро, яке від неї відходить. Це ребро вже проходиться повторно.

Ребро – $[6; 8]$

$S = [1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8, 4, 7, 6, 8]$

$C = [1, 2, 5, 2]$

Список ребер, що залишилися:

$[]$

Список повторних ребер:

$[]$

Крок 26.

Так як списки ребер пусті, це означає, що було пройдено усі ребра, які були потрібні, а отже алгоритм закінчено.

Тепер потрібно склеїти списки S і C . Це і буде обходом листоноші.

[1, 2, 4, 5, 7, 8, 1, 3, 6, 7, 3, 8, 4, 7, 6, 8, 2, 5, 2, 1]

Загальна сума ваг пройдених рівна 1227

Програмна реалізація (python):

```
def Chinese_Postman(graph):
    odds = get_odd(graph)
    if(len(odds)==0):
        return sum_edges(graph)
    pairs = gen_pairs(odds)
    l = (len(pairs)+1)//2

    pairings_sum = []

    def get_pairs(pairs, done = [], final = []):

        if(pairs[0][0][0] not in done):
            done.append(pairs[0][0][0])

            for i in pairs[0]:
                f = final[:]
                val = done[:]
                if(i[1] not in val):
                    f.append(i)
                else:
                    continue

            if(len(f)==l):
                pairings_sum.append(f)
                return
            else:
                val.append(i[1])
                get_pairs(pairs[1:], val, f)

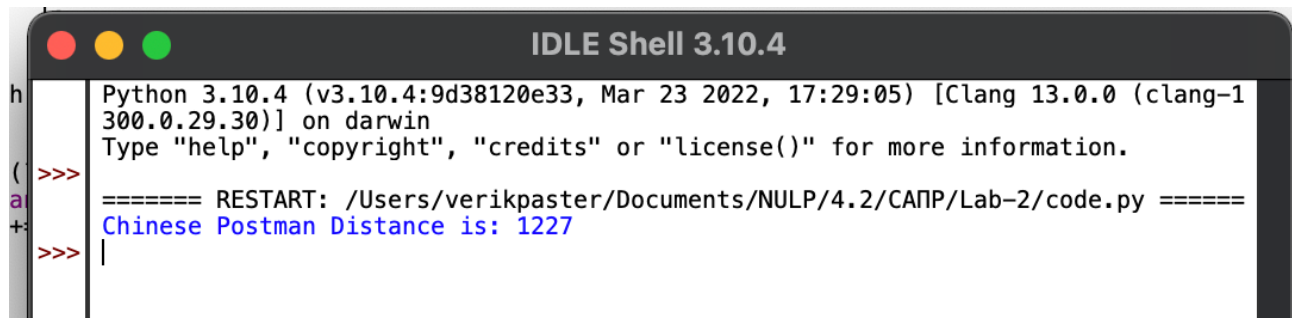
        else:
            get_pairs(pairs[1:], done, final)

    get_pairs(pairs)
    min_sums = []

    for i in pairings_sum:
        s = 0
        for j in range(len(i)):
            s += dijktra(graph, i[j][0], i[j][1])
        min_sums.append(s)

    added_dis = min(min_sums)
    chinese_dis = added_dis + sum_edges(graph)
    return chinese_dis
```

Результати виконання програми



```
Python 3.10.4 (v3.10.4:9d38120e33, Mar 23 2022, 17:29:05) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/verikpaster/Documents/NULP/4.2/CAПP/Lab-2/code.py =====
Chinese Postman Distance is: 1227
>>>
```

Висновок:

В ході даної лабораторної роботи я ознайомилась з поняттями Ейлерів граф, Ейлерів маршрут та Ейлерів цикл, також на основі цих понять відбулося вивчення алгоритму листоноші. Було проведені ручні обчислення для обходу графа, які відображені в звіті, а також реалізовано програму. Результати програми і ручних обчислень зійшлися.