**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Звіт**

з лабораторної роботи № 8

„ **Евристичні алгоритми** ”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-21 Скрипець Ольга Олександрівна*

**Перевірив**

(прізвище, ім'я, по батькові)

Київ 2023

**Зміст**

[1 Мета лабораторної роботи 3](#_Toc136935675)

[2 Завдання 4](#_Toc136935676)

[3 Виконання 5](#_Toc136935677)

[3.1 Аналіз часової складності 5](#_Toc136935678)

[3.2 Програмна реалізація алгоритму 6](#_Toc136935679)

[3.2.1 Вихідний код 6](#_Toc136935680)

[3.2.2 Приклади роботи 9](#_Toc136935681)

[3.3.3. Висновок 10](#_Toc136935682)

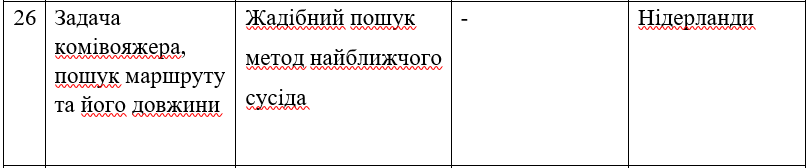
# 1 Мета лабораторної роботи

Мета роботи – вивчити основні підходи формалізації евристичних алгоритмів і вирішення типових задач з їх допомогою.

# 2 Завдання

Для **задачі комівояжера**, вибрати 15 міст в країні згідно варіанту (таблиця 2.1) і записати для них найкоротшу відстань по дорозі, у випадку прямого сполучення між ними. Для визначення відстані рекомендується використовувати інтернет сервіси (наприклад Google Maps).

Відповідь вивести у вигляді (Місто1-Місто2 Відстань: 234км Маршрут: Місто1 → Місто3 → Місто4 → Місто2). **Вивести кожну пару міст, для обох алгоритмів.**



* 1. Програмна реалізація

## 2.1.1 Вихідний код

def find\_minimum\_route(distances, city\_names):

num\_cities = len(distances)

visited = [False] \* num\_cities

route = []

current\_city = 0 # Початкове місто (Амстердам)

total\_distance = 0

visited[current\_city] = True

route.append(current\_city)

while len(route) < num\_cities:

nearest\_city = None

nearest\_distance = float('inf')

for city in range(num\_cities):

if not visited[city] and distances[current\_city][city] < nearest\_distance:

nearest\_city = city

nearest\_distance = distances[current\_city][city]

if nearest\_city is None:

break

visited[nearest\_city] = True

route.append(nearest\_city)

total\_distance += nearest\_distance

current\_city = nearest\_city

# Додавання повернення до початкового міста

route.append(0) # Початкове місто

total\_distance += distances[current\_city][0]

return route, total\_distance

# Таблиця відстаней між містами

"""distances = [

[0, 78, 64, 48, 122, 111, 32, 181, 105, 115, 88, 32, 161, 98, 53],

[78, 0, 24, 61, 110, 77, 101, 246, 51, 109, 124, 73, 197, 113, 84],

[64, 24, 0, 68, 143, 108, 87, 236, 82, 136, 131, 60, 204, 119, 88],

[48, 61, 68, 0, 91, 81, 41, 187, 74, 90, 68, 59, 141, 65, 28],

[122, 110, 143, 91, 0, 39, 127, 253, 59, 66, 115, 139, 180, 84, 112],

[111, 77, 108, 81, 39, 0, 115, 262, 28, 81, 130, 128, 194, 99, 100],

[32, 101, 87, 41, 127, 115, 0, 154, 109, 112, 77, 54, 150, 87, 38],

[181, 246, 236, 187, 253, 262, 154, 0, 255, 206, 144, 203, 148, 173, 175],

[105, 51, 82, 74, 59, 28, 109, 255, 0, 95, 133, 121, 206, 114, 93],

[115, 109, 136, 90, 66, 81, 112, 206, 95, 0, 69, 133, 134, 24, 81],

[88, 124, 131, 68, 115, 130, 77, 144, 133, 69, 0, 110, 75, 36, 48],

[32, 73, 60, 59, 139, 128, 54, 203, 121, 133, 110, 0, 182, 115, 70],

[161, 197, 204, 141, 180, 194, 150, 148, 206, 134, 75, 182, 0, 99, 121],

[98, 113, 119, 65, 84, 99, 87, 173, 114, 24, 36, 115, 99, 0, 56],

[53, 84, 88, 28, 112, 100, 38, 175, 93, 81, 48, 70, 121, 56, 0]

]

"""

distances = [

[0, 78, 64, 48, 122, 111, 32, 181, 105, 115, 88, 32],

[78, 0, 24, 61, 110, 77, 101, 246, 51, 109, 124, 73],

[64, 24, 0, 68, 143, 108, 87, 236, 82, 136, 131, 60],

[48, 61, 68, 0, 91, 81, 41, 187, 74, 90, 68, 59],

[122, 110, 143, 91, 0, 39, 127, 253, 59, 66, 115, 139],

[111, 77, 108, 81, 39, 0, 115, 262, 28, 81, 130, 128],

[32, 101, 87, 41, 127, 115, 0, 154, 109, 112, 77, 54],

[181, 246, 236, 187, 253, 262, 154, 0, 255, 206, 144, 203],

[105, 51, 82, 74, 59, 28, 109, 255, 0, 95, 133, 121],

[115, 109, 136, 90, 66, 81, 112, 206, 95, 0, 69, 133],

[88, 124, 131, 68, 115, 130, 77, 144, 133, 69, 0, 110],

[32, 73, 60, 59, 139, 128, 54, 203, 121, 133, 110, 0]

]

# Масив з назвами міст

city\_names = [

"Амстердам", "Роттердам", "Гаага", "Утрехт", "Ейндговен", "Тілбург", "Алмере", "Гронінген", "Бреда",

"Неймеген", "Апелдорн", "Гарлем", "Енсхеде", "Арнем", "Амерсфорт"

]

def format\_route(route, city\_names):

formatted\_route = 'Маршрут: ' + ' → '.join(city\_names[city] for city in route) + f', Довжина: {total\_distance}км'

return formatted\_route

# Знаходження мінімального маршруту

route, total\_distance = find\_minimum\_route(distances, city\_names)

# Форматування виводу

formatted\_route = format\_route(route, city\_names)

print(formatted\_route)

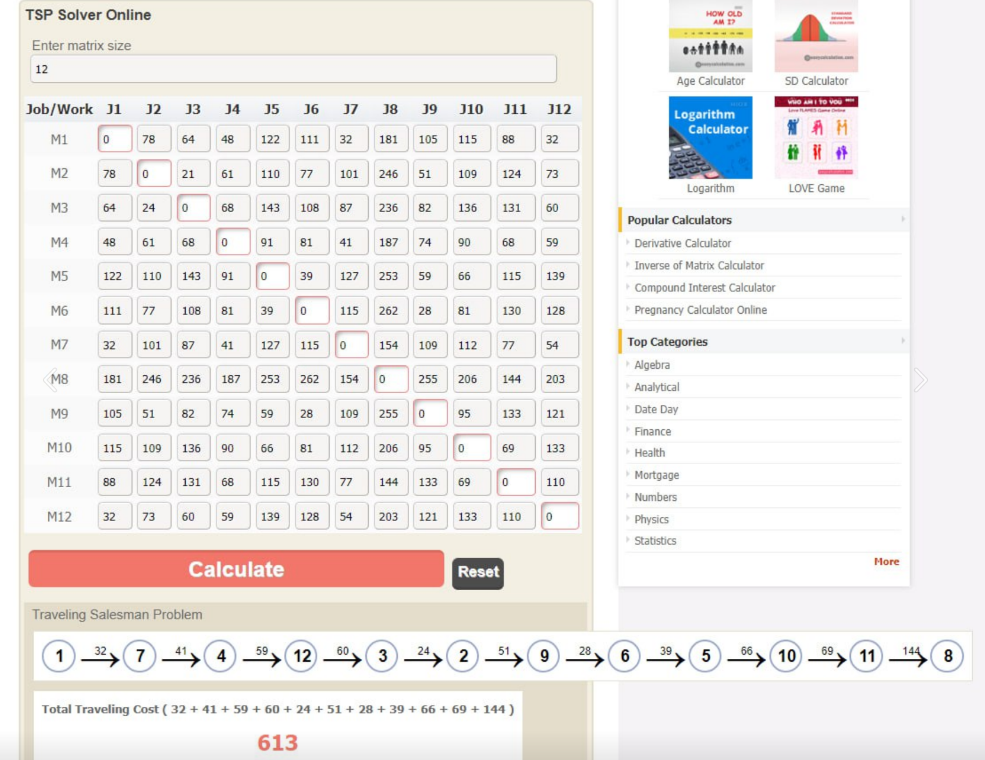
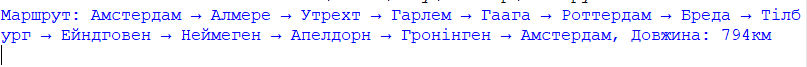
2.2 Вхідні дані задачі

У таблиці 2.1 наведені відстані між містами по дорозі, якщо між ними є прямий шлях.

Таблиця 2.1 – Відстань між містами по дорозі

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Амстердам | Роттердам | Гаага | Утрехт | Ейндговен | Тілбург | Алмере | Гронінген | Бреда | Неймеген | Апелдорн | Гарлем | Енсхеде | Арнем | Амерсфорт |
| Амстердам | 0 | 78 км | 64 км | 48 км | 122 км | 111 км | 32 км | 181 км | 105 км | 115 км | 88 км | 32 км | 161 км | 98 км | 53 км |
| Роттердам | 78 км | 0 | 24 км | 61 км | 110 км | 77 км | 101 км | 246 км | 51 км | 109 км | 124 км | 73 км | 197 км | 113 км | 84 км |
| Гаага | 64 км | 24 км | 0 | 68 км | 143 км | 108 км | 87 км | 236 км | 82 км | 136 км | 131 км | 60 км | 204 км | 119 км | 88 км |
| Утрехт | 48 км | 61 км | 68 км | 0 | 91 км | 81 км | 41 км | 187 км | 74 км | 90 км | 68 км | 59 км | 141 км | 65 км | 28 км |
| Ейндговен | 122 км | 110 км | 143 км | 91 км | 0 | 39 км | 127 км | 253 км | 59 км | 66 км | 115 км | 139 км | 180 км | 84 км | 112 км |
| Тілбург | 111 км | 77 км | 108 км | 81 км | 39 км | 0 | 115 км | 262 км | 28 км | 81 км | 130 км | 128 км | 194 км | 99 км | 100 км |
| Алмере | 32 км | 101 км | 87 км | 41 км | 127 км | 115 км | 0 | 154 км | 109 км | 112 км | 77 км | 54 км | 150 км | 87 км | 38 км |
| Гронінген | 181 км | 246 км | 236 км | 187 км | 253 км | 262 км | 154 км | 0 | 255 км | 206 км | 144 км | 203 км | 148 км | 173 км | 175 км |
| Бреда | 105 км | 51 км | 82 км | 74 км | 59 км | 28 км | 109 км | 255 км | 0 | 95 км | 133 км | 121 км | 206 км | 114 км | 93 км |
| Неймеген | 115 км | 109 км | 136 км | 90 км | 66 км | 81 км | 112 км | 206 км | 95 км | 0 | 69 км | 133 км | 134 км | 24 км | 81 км |
| Апелдорн | 88 км | 124 км | 131 км | 68 км | 115 км | 130 км | 77 км | 144 км | 133 км | 69 км | 0 | 110 км | 75 км | 36 км | 48 км |
| Гарлем | 32 км | 73 км | 60 км | 59 км | 139 км | 128 км | 54 км | 203 км | 121 км | 133 км | 110 км | 0 | 182 км | 115 км | 70 км |
| Енсхеде | 161 км | 197 км | 204 км | 141 км | 180 км | 194 км | 150 км | 148 км | 206 км | 134 км | 75 км | 182 км | 0 | 99 км | 121 км |
| Арнем | 98 км | 113 км | 119 км | 65 км | 84 км | 99 км | 87 км | 173 км | 114 км | 24 км | 36 км | 115 км | 99 км | 0 | 56 км |
| Амерсфорт | 53 км | 84 км | 88 км | 28 км | 112 км | 100 км | 38 км | 175 км | 93 км | 81 км | 48 км | 70 км | 121 км | 56 км | 0 |

2.3 Тестування



ВИСНОВОК

У даній лабораторній роботі була розроблена програма для вирішення задачі комівояжера методом найближчого сусіда. Метою було знайти маршрут мінімальної довжини, який включає всі міста.

Алгоритм найближчого сусіда працює за принципом вибору найближчого непройденого міста на кожному кроці. Починаючи з початкового міста, на кожному кроці вибирається найближче непройдене місто і додається до маршруту. Цей процес повторюється до тих пір, поки всі міста не будуть включені в маршрут.

Складність цього алгоритму залежить від кількості міст. У найгіршому випадку, коли кількість міст дорівнює n, алгоритм має складність O(n^2). Така складність виникає через необхідність обчислення відстані між кожною парою міст. Оскільки на кожному кроці потрібно знайти найближче місто, а це вимагає порівняння з усіма непройденими містами, час виконання залежить квадратично від кількості міст.

Алгоритм найближчого сусіда є простим та легко реалізовується. Однак, його головний недолік полягає в тому, що він не завжди знаходить оптимальний маршрут. Він може дати задовільний результат для невеликої кількості міст, але для великих задач його розв'язок може бути далеким від оптимального.

У цілому, лабораторна робота дозволила ознайомитись з методом найближчого сусіда для розв'язання задачі комівояжера. Вона також показала нам, що вибір певного алгоритму залежить від специфіки задачі та вимог до результату.