**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе № 2**

**по дисциплине «Построение и Анализ Алгоритмов»**

Тема: «**Коммивояжёр**»

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Коршков А.А. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2025

# Цель работы

Изучить различные алгоритмы для решения задачи коммивояжёра. Написать программу решения коммивояжёра через динамическое программирование (точный метод: итеративная реализация) и через АЛШ-2 (алгоритм лучшего соседа через МОД (минимальное остовное дерево)) с возможностью генерации матрицы весов (симметричной/несимметричной), сохранения её в файл и использования в программе в качестве входных данных.

# Задание

**Динамическое программирование**

Напишите программу, решающую задачу коммивояжера. Нужно найти кратчайший маршрут, который проходит через все заданные города ровно один раз и возвращается в исходный город. Не все города могут быть напрямую связаны друг с другом.

**Входные данные:**

* n - количество городов (5 ≤ n ≤ 15).
* Матрица расстояний между городами размером n×n, где graph[i][j] обозначает расстояние от города i до города j. Если graph[i][j]=0 (и i ≠ j), это означает, что прямого пути между городами нет.

**Выходные данные:**

* Минимальная стоимость маршрута, проходящего через все города и возвращающегося в начальный город.
* Оптимальный путь в виде последовательности посещаемых городов, начинающейся и заканчивающейся в начальном городе.
* Если такого пути не существует, вывести "no path".

**Sample Input 1:**

5

0 1 13 23 7

12 0 15 18 28

21 29 0 33 28

23 19 34 0 38

5 40 7 39 0

**Sample Output 1:**

78

0 4 2 3 1 0

**Sample Input 2:**

3

0 1 0

1 0 1

0 1 0

**Sample Output 2:**

no path

**Задание варианта:**

**№8.** Точный метод: динамическое программирование (не МВиГ), итеративная реализация.

Приближённый алгоритм: АЛШ-2.

**Примечания для варианта:**

Требование перед сдачей: прохождение кода в задании 3.1 на Stepik.

Замечание к варианту 8 АЛШ-2 начинать со стартовой вершины.

## Основные теоретические положения

**Описание алгоритмов:**

Динамическое программирование. Итеративная реализация. Алгоритм Хельда-Капра.

Алгоритм лучшего соседа. Решение через

**Сложность алгоритмов:**

Итеративная реализация (алгоритма Хельда-Капра):

Сложность по времени:

Сложность по памяти:

Приближённый алгоритм лучшего соседа через МОД:

Сложность по времени:

Сложность по памяти:

## Выполнение работы

**Описание работы**

Был создан parser.py содержащий функцию get\_args() -> Namespace, которая возвращает аргументы командной строки через модуль argparse. В нём есть флаги для запуска генерации матрицы (-g, --generate) , настройки количества городов (-c, --count), максимального веса ребра (--max-weight), симметричности матрицы (-s), файл, куда нужно сохранить сгенерированную матрицу (-o, --output), файл, откуда получить информацию о матрице (-i, --input), метод решения задачи (--method {exact, approx}) и вывод справки.

Файл loader.py содержит функции для генерации, сохранения и загрузки матрицы.

generate\_mx(n: int, symmetric: bool = *False*, max\_weight: int = 100) -> list[list[int]] – генерирует матрицу размера n, симметричную/несимметричную, с заданным максимальным размером ребра.

load\_mx(file\_name: str) -> tuple[int, list[list[int]]] – загружает матрицу весов из файла, возвращает количество городов и саму матрицу.

write\_mx(file\_name: str, weight\_matrix: list[list[int]]) -> *None* – сохраняет матрицу в заданный файл

Файл tsp\_exact.py содержит функцию tsp\_dp(n: int, graph: list[list[int]]) -> *None* для решения задачи коммивояжёра методом динамического программирования (алгоритм Хельда-Капра).

Файл tsp\_approx.py содержит функции для решения задачи коммивояжёра приближённым алгоритмом АЛШ-2 (Алгоритм лучшего соседа через оставное дерево).

prim\_mst\_cost(graph, nodes) -> int | float - Функция для вычисления стоимости минимального остовного дерева с использованием алгоритма Прима.

# tsp\_alsh2(n: int, graph: list[list[int]]) -> None – решение задачи через АЛШ-2

В main.py есть функция main() -> *None*

# Тестирование

Таблица 1 – Тестирование алгоритмов

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Выходные данные | Комментарии |
| 1 | 5  0 1 13 23 7  12 0 15 18 28  21 29 0 33 28  23 19 34 0 38  5 40 7 39 0 | 78  0 4 2 3 1 0 | Точный метод. Матрица несимметричная. Результат вычислен корректно |
| 2 | 3  0 1 0  1 0 1  0 1 0 | no path | Точный метод. Пути нет. |
| 3 | 6  0 29 16 8 33 39  29 0 11 24 9 13  16 11 0 39 29 13  8 24 39 0 18 50  33 9 29 18 0 50  39 13 13 50 50 0 | 77  0 3 4 1 5 2 0 | Точный метод. Матрица симметричная. |
| 4 | 5  0 1 13 23 7  12 0 15 18 28  21 29 0 33 28  23 19 34 0 38  5 40 7 39 0 | 84  0 4 2 1 3 0 | Приближённый алгоритм. Матрица несимметричная. В отличии от точного метода конечная стоимость выше, но результат был получен быстрее.  Количество пройденных городов и сами города не изменились, однако порядок городов изменился. |
| 5 | 3  0 1 0  1 0 1  0 1 0 | no path | Приближённый алгоритм. Пути нет. |
| 6 | 6  0 29 16 8 33 39  29 0 11 24 9 13  16 11 0 39 29 13  8 24 39 0 18 50  33 9 29 18 0 50  39 13 13 50 50 0 | 98  0 3 4 1 2 5 0 | Приближённый алгоритм. Матрица симметричная. Как и в случае 4 алгоритм отработал быстро и с большей стоимостью.  Порядок городов изменён. |

# Выводы

Был реализован алгоритм, решающий алгоритм Коммивояжёра через динамическое программирование с итеративным подходом и через приближённый алгоритм лучшего соседа через МОД. Также написаны функции для генерации, загрузки и сохранения матрицы весов.

ПРИЛОЖЕНИЕ А  
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py