**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Построение и анализ алгоритмов»**

Тема: Задача Коммивояжера

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Коняева М. В. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2025

**Цель работы.**

Изучить принцип работы алгоритмов нахождения пути коммивояжера на графах.

**Задание.**

Группа вариантов 1-5:

Решить ЗК двумя методами в соответствии с вариантом: 1) Методом ВиГ. 2)Приближённым методом.

Дано: матрица весов графа, все веса неотрицательны; стартовая вершина.

Найти: путь коммивояжёра (последовательность вершин) и его стоимость.

Вариант 1.

МВиГ: Алгоритм Литтла.

Приближённый алгоритм: 2-приближение по МОД (АДО МОД).

Замечание к варианту 1. АДО МОД является 2-приближением только для евклидовой матрицы. Начинать обход МОД со стартовой вершины.

Замечание к алгоритму Литтла.

При каждом выборе дуги́ XY для ветвления получаем левую ветвь, в которой включена дуга XY и запрещена некоторая дуга Y'X' (правую ветвь, в которой запрещена дуга XY, сейчас не рассматриваем). Обратите внимание, что если выбрана дуга AB, то не обязательно запрещённой оказывается дуга BA. Пока не было шага по левой ветви, выбор дуги AB действительно приводит к запрещению дуги BA. Если же шаг по левой ветви уже был, то не всегда запретить надо обратную дугу. Например:

- Уже выбрана дуга AB, выбирается дуга BC —> запретить надо CA.

- Уже выбраны дуги AB и CD, выбирается дуга BC —> запретить надо DA.

**Описание алгоритма.**

Описание алгоритма Литтла.

Алгоритм Литла предназначен для решения задачи коммивояжера с использованием метода ветвей и границ. Входные данные представляют собой матрицу смежности графа.

Сначала выполняется редукция матрицы: для каждой строки находится минимальный элемент, который затем вычитается из всех элементов этой строки. Аналогичная операция выполняется для столбцов. Далее ищется “тяжёлый ноль” — нулевой элемент, у которого сумма минимальных значений в его строке и столбце максимальна.

На этом этапе происходит ветвление. В одной ветви из матрицы исключаются строка и столбец, содержащие выбранный ноль, а также запрещается движение, которое могло бы образовать цикл. В другой ветви этот ноль заменяется на бесконечность (∞). Стоимость каждой ветви фиксируется.

Когда находится первое допустимое решение, оно принимается за минимально возможное. В дальнейшем ветви с заведомо большим значением отсекаются. Если в процессе вычислений обнаруживается более выгодный маршрут, он заменяет текущий оптимальный. Такой подход позволяет избежать полного перебора всех возможных вариантов, обеспечивая нахождение оптимального решения.

Рекурсивный вызов метода branch: среднее количество узлов в дереве оценивается как O(c^n), где c < 2 (обычно c ≈ 1.26), но точное значение зависит от структуры матрицы). На каждом уровне рекурсии выполняются операции поиска нулевого элемента и редукции матрицы, каждая из которых имеет сложность O(n^2)

Следовательно, сложность рекурсивного вызова метода составляет O(c^n \* n^2), а общая сложность алгоритма также равна O(c^n \* n^2).

Оценка сложности по памяти:  
В худшем случае рекурсивный вызов метода branch приводит к полному обходу бинарного дерева, в котором число узлов приблизительно равно O(c^n), где n— количество вершин в графе. На каждом уровне рекурсии создается копия матрицы смежности, занимающая O(n^2) памяти. Таким образом, общая сложность по памяти составляет O(c^n \* n^2)

Описание алгоритма АДО МОД.

Алгоритм решения задачи коммивояжёра с использованием минимального остовного дерева (MST) начинается с построения MST с помощью алгоритма Прима. Сначала инициализируются массивы для хранения минимальных весов рёбер, родительских вершин и информации о том, включена ли вершина в MST. Начинаем с первой вершины, присваиваем ей минимальный вес 0 и помещаем её в приоритетную очередь. Затем, пока очередь не пуста, извлекаем вершины с минимальным весом и добавляем их в MST. Для каждой соседней вершины, если её ещё не добавили в MST и если вес рёбер меньше текущего минимального, обновляем её вес и добавляем в очередь с новым весом. После завершения этого этапа у нас есть минимальное остовное дерево, представляемое списком рёбер, соединяющих все вершины графа.

Затем, на основе полученных рёбер MST, строится список смежности для каждой вершины. После этого выполняется обход этого дерева в глубину (DFS). Для этого создаём стек и список посещённых вершин. Начинаем обход с заданной стартовой вершины, добавляем её в стек и продолжаем обход до тех пор, пока стек не станет пустым. Каждую посещённую вершину добавляем в путь обхода и отмечаем как посещённую. Соседей текущей вершины добавляем в стек в порядке возрастания веса рёбер. В результате мы получаем путь, который является обходом всех вершин в MST.

После выполнения обхода замыкаем путь, добавляя стартовую вершину в конец, чтобы получить полный цикл. Затем вычисляем суммарную стоимость маршрута, проходя по всем рёбрам пути и суммируя их веса. Если между двумя вершинами нет ребра, путь невозможен, и алгоритм сообщает об ошибке. В конце возвращается общая стоимость маршрута и сам путь, который является приближённым решением задачи коммивояжёра.

Сложность по времени алгоритма:

**Алгоритм Прима:**

* Сортировка рёбер: не требуется, так как используется приоритетная очередь для выбора минимальных рёбер из ограниченного множества.
* Приоритетная очередь (для выбора минимальных рёбер): O(logV) для каждой операции вставки и извлечения, что даёт O(ElogV) для всех рёбер.
* Создание множества вершин: O(V)
* Общий результат Прима: O(ElogV).
* Обход DFS по MST: O(V+E).
* Общая сложность: O(ElogV).

**Сложность по памяти алгоритма:**

* Матрица смежности (входной граф): O(V^2)
* Список рёбер (используется в алгоритме Прима): O(E)
* Приоритетная очередь: O(V)
* Хранение MST (в виде списка смежности): O(V+E)
* Структуры DFS (массив посещённых вершин, стек рекурсии): O(V)
* Общая сложность по памяти: O(V^2)

**Описание функций и структур данных.**

Класс Matrix представляет собой структуру данных для хранения матрицы стоимостей (например, для задачи коммивояжера или аналогичных задач). В нем определены следующие функции:

1. **Конструктор Matrix(int size)**: Инициализирует квадратную матрицу заданного размера, заполняя все элементы нулями.
2. **Конструктор Matrix(std::vector<std::vector<double>> matrix)**: Инициализирует матрицу с заранее заданными значениями.
3. **Конструктор по умолчанию Matrix()**: Пустой конструктор.
4. **setCell(int row, int col, int val)**: Устанавливает значение в ячейку матрицы в строке row и столбце col на значение val.
5. **getCell(int row, int col)**: Возвращает значение в ячейке матрицы по индексу строки row и столбца col.
6. **getSize()**: Возвращает размерность матрицы.
7. **deleteRowCol(int row, int col)**: Удаляет строку и столбец, заменяя все элементы в строке и столбце на INF, чтобы они не учитывались.
8. **getMinInRow(int row)**: Находит минимальный элемент в строке row.
9. **getMinInCol(int col)**: Находит минимальный элемент в столбце col.
10. **print()**: Выводит матрицу в консоль для отладки, заменяя значения INF на -1 для удобства отображения.
11. **generate(bool is\_symmetric = false, int min = 1, int max = 100)**: Генерирует случайную матрицу. Если is\_symmetric равно true, матрица будет симметричной (элементы выше главной диагонали будут зеркально отображены относительно главной диагонали).
12. **readMatrixFromConsole()**: Считывает матрицу с консоли, заменяя все -1 на INF.
13. **saveToFile(const std::string& filename)**: Сохраняет матрицу в файл с заданным именем filename.
14. **loadFromFile(const std::string& filename)**: Загружает матрицу из файла с заданным именем filename.

Реализован класс BranchAndBound для решения задачи коммивояжера методом ветвей и границ (Branch and Bound).

1. **reduce(Matrix& matrix):** Эта функция выполняет редукцию матрицы путём вычитания минимальных значений из каждой строки и каждого столбца. Для каждой строки из неё вычитается минимальный элемент, а для каждого столбца — минимальный элемент. Результатом является обновлённая матрица и вычисленная сумма сокращений, которая добавляется к общей стоимости.
2. **calcBestArc(Matrix& matrix):** Функция находит наилучшую дугу (рёбро) для дальнейшего включения в путь. Она ищет все нулевые элементы в матрице (кандидаты на выбор рёбер), затем вычисляет для каждого из них стоимость отказа (сумма минимальных элементов в строках и столбцах, в которых находятся эти нули). Выбирается дуга с максимальной стоимостью отказа.
3. **calcForbiddenArc(Matrix& matrix, std::vector<std::pair<int, int>>& path, int from, int to):** Эта функция определяет запрещённую дугу для пути. Для данного пути (последовательности рёбер) она находит все рёбра, которые ведут в вершину X и из вершины Y, и запрещает обратную дугу (например, если из вершины W есть ребро к X и из вершины Y выходит ребро к Z, то запрещается дуга Z -> W).
4. **calcBnB(Matrix& matrix, int cost, std::vector<std::pair<int, int>>& path, std::vector<std::pair<int, int>>& bestPath, int& bestCost):** Это рекурсивная функция, которая реализует сам алгоритм ветвей и границ. На каждом шаге она выполняет редукцию матрицы, вычисляет новую стоимость пути и, если эта стоимость не превышает наилучшего найденного решения, выбирает наилучшее ребро для продолжения пути. Функция работает с двумя ветвями: первой, когда добавляется новое ребро в путь, и второй, когда ребро исключается.
5. **formWay(std::vector<std::pair<int, int>>& arcs, int start\_vertex):** Эта функция восстанавливает путь из списка рёбер. Двигаясь по рёбрам вперёд и назад, она формирует последовательность вершин, представляющую путь от начальной вершины до конечной.

Реализован класс ModTSP, который решает задачу коммивояжёра с использованием метода минимального остовного дерева (MST). Описание основных функций класса:

1. **primMST(Matrix& matrix)**: Эта функция строит минимальное остовное дерево (MST) с помощью алгоритма Прима. На вход она получает матрицу, представляющую граф, а на выходе возвращает список рёбер MST. В процессе работы она использует приоритетную очередь для выбора наименьшего веса рёбер, чтобы добавить вершины в MST. Каждое добавление вершины сопровождается выводом информации о добавленных рёбрах и вершинах.
2. **dfsMST(const std::vector<std::vector<std::pair<int, double>>>& mstAdj, int start\_vertex)**: Эта функция выполняет обход минимального остовного дерева (MST) в глубину (DFS), начиная с указанной вершины. Она возвращает список вершин в порядке их посещения. Сначала в стек добавляется стартовая вершина, затем рекурсивно посещаются все соседние вершины. Соседи сортируются по весу рёбер, чтобы обход происходил в порядке увеличения веса.
3. **approximateTSP(Matrix& matrix, int start\_vertex)**: Эта функция находит приближённое решение задачи коммивояжёра с использованием минимального остовного дерева. Алгоритм включает в себя несколько шагов:
   * Строится минимальное остовное дерево (MST) с помощью алгоритма Прима.
   * Создаётся список смежности для этого дерева.
   * Выполняется обход дерева в глубину (DFS), начиная с указанной стартовой вершины.
   * Замыкается путь, добавляя стартовую вершину в конец маршрута.
   * Рассчитывается суммарная стоимость маршрута. Функция возвращает пару: суммарную стоимость маршрута и сам путь (список вершин в порядке посещения).

**Тестирование.**

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Выходные данные | Комментарий |
| 1 | Алгоритм Литтла  3  -1 1 3  3 -1 1  1 2 -1 | Best Path: 0 1 2  Cost: 3.0 | Верно |
| 2 | Алгоритм Литтла  4  -1 3 4 1  1 -1 3 4  9 2 -1 4  8 9 2 -1 | Best Path: 0 3 2 1  Cost: 6.0 | Верно |
| 3 | Алгоритм АДО МОД  2 -1 18.97 22.36 19.42 3.61 18.97 -1 35.61 38.01 17.0 22.36 35.61 -1 16.28 21.19 19.42 38.01 16.28 -1 21.02 3.61 17.0 21.19 21.02 -1 | Path: 2 3 0 4 1 2  Cost: 91.92 | Верно |
| 4 | Алгоритм АДО МОД 0  2147483647 20 2 6 16  20 2147483647 9 4 19  2 9 2147483647 16 19  6 4 16 2147483647 6  16 19 19 6 2147483647 | Path: 0 3 4 1 2 0  Cost: 42 | Верно |

Вывод с комментариями для 4 теста.  
Введите строку template (образец): dsds

Введите строку text (текст): dsdsdfdsdsfdssd

Загруженная матрица:

[inf, 20, 2, 6, 16]

[20, inf, 9, 4, 19]

[2, 9, inf, 16, 19]

[6, 4, 16, inf, 6]

[16, 19, 19, 6, inf]

Метод Литтла:

=============================

Текущий путь: [], Текущий город: 0, Текущая нижняя оценка: 0

Редукция матрицы:

Матрица до редукции:

[inf, 20, 2, 6, 16]

[20, inf, 9, 4, 19]

[2, 9, inf, 16, 19]

[6, 4, 16, inf, 6]

[16, 19, 19, 6, inf]

Из строки 0 вычли 2

Из строки 1 вычли 4

Из строки 2 вычли 2

Из строки 3 вычли 4

Из строки 4 вычли 6

Из столбца 0 ничего не вычли

Из столбца 1 ничего не вычли

Из столбца 2 ничего не вычли

Из столбца 3 ничего не вычли

Из столбца 4 вычли 2

Матрица после редукции:

[inf, 18, 0, 4, 12]

[16, inf, 5, 0, 13]

[0, 7, inf, 14, 15]

[2, 0, 12, inf, 0]

[10, 13, 13, 0, inf]

Общее сокращение: 20.0

Нижняя граница стоимости: 20.0, Текущая лучшая стоимость: inf

Для элемента [0][2] штраф 9

Для элемента [1][3] штраф 5

Для элемента [2][0] штраф 9

Для элемента [3][1] штраф 7

Для элемента [3][4] штраф 12

Для элемента [4][3] штраф 10

Выбрано ребро 3 -> 4 с максимальным коэффициентом 12

Ветка 1: Включаем ребро 3 -> 4, его вес: 6

Исключаем ребро 4 -> 3

=============================

Текущий путь: [(3, 4)], Текущий город: 4, Текущая нижняя оценка: 20.0

Редукция матрицы:

Матрица до редукции:

[inf, 18, 0, 4, inf]

[16, inf, 5, 0, inf]

[0, 7, inf, 14, inf]

[inf, inf, inf, inf, inf]

[10, 13, 13, inf, inf]

Из строки 0 ничего не вычли

Из строки 1 ничего не вычли

Из строки 2 ничего не вычли

Из строки 3 ничего не вычли

Из строки 4 вычли 10

Из столбца 0 ничего не вычли

Из столбца 1 вычли 3

Из столбца 2 ничего не вычли

Из столбца 3 ничего не вычли

Из столбца 4 ничего не вычли

Матрица после редукции:

[inf, 15, 0, 4, inf]

[16, inf, 5, 0, inf]

[0, 4, inf, 14, inf]

[inf, inf, inf, inf, inf]

[0, 0, 3, inf, inf]

Общее сокращение: 13.0

Нижняя граница стоимости: 33.0, Текущая лучшая стоимость: inf

Для элемента [0][2] штраф 7

Для элемента [1][3] штраф 9

Для элемента [2][0] штраф 4

Для элемента [4][0] штраф 0

Для элемента [4][1] штраф 4

Выбрано ребро 1 -> 3 с максимальным коэффициентом 9

Ветка 1: Включаем ребро 1 -> 3, его вес: 4

Исключаем ребро 4 -> 1

=============================

Текущий путь: [(3, 4), (1, 3)], Текущий город: 3, Текущая нижняя оценка: 33.0

Редукция матрицы:

Матрица до редукции:

[inf, 15, 0, inf, inf]

[inf, inf, inf, inf, inf]

[0, 4, inf, inf, inf]

[inf, inf, inf, inf, inf]

[0, inf, 3, inf, inf]

Из строки 0 ничего не вычли

Из строки 1 ничего не вычли

Из строки 2 ничего не вычли

Из строки 3 ничего не вычли

Из строки 4 ничего не вычли

Из столбца 0 ничего не вычли

Из столбца 1 вычли 4

Из столбца 2 ничего не вычли

Из столбца 3 ничего не вычли

Из столбца 4 ничего не вычли

Матрица после редукции:

[inf, 11, 0, inf, inf]

[inf, inf, inf, inf, inf]

[0, 0, inf, inf, inf]

[inf, inf, inf, inf, inf]

[0, inf, 3, inf, inf]

Общее сокращение: 4.0

Нижняя граница стоимости: 37.0, Текущая лучшая стоимость: inf

Для элемента [0][2] штраф 14

Для элемента [2][0] штраф 0

Для элемента [2][1] штраф 11

Для элемента [4][0] штраф 3

Выбрано ребро 0 -> 2 с максимальным коэффициентом 14

Ветка 1: Включаем ребро 0 -> 2, его вес: 2

Исключаем ребро 2 -> 0

=============================

Текущий путь: [(3, 4), (1, 3), (0, 2)], Текущий город: 2, Текущая нижняя оценка: 37.0

Редукция матрицы:

Матрица до редукции:

[inf, inf, inf, inf, inf]

[inf, inf, inf, inf, inf]

[inf, 0, inf, inf, inf]

[inf, inf, inf, inf, inf]

[0, inf, inf, inf, inf]

Из строки 0 ничего не вычли

Из строки 1 ничего не вычли

Из строки 2 ничего не вычли

Из строки 3 ничего не вычли

Из строки 4 ничего не вычли

Из столбца 0 ничего не вычли

Из столбца 1 ничего не вычли

Из столбца 2 ничего не вычли

Из столбца 3 ничего не вычли

Из столбца 4 ничего не вычли

Матрица после редукции:

[inf, inf, inf, inf, inf]

[inf, inf, inf, inf, inf]

[inf, 0, inf, inf, inf]

[inf, inf, inf, inf, inf]

[0, inf, inf, inf, inf]

Общее сокращение: 0.0

Нижняя граница стоимости: 37.0, Текущая лучшая стоимость: inf

Для элемента [2][1] штраф 0

Для элемента [4][0] штраф 0

Выбрано ребро 2 -> 1 с максимальным коэффициентом 0

Ветка 1: Включаем ребро 2 -> 1, его вес: 9

Исключаем ребро 4 -> 0

=============================

Текущий путь: [(3, 4), (1, 3), (0, 2), (2, 1)], Текущий город: 1, Текущая нижняя оценка: 37.0

Включенные ребра: [(3, 4), (1, 3), (0, 2), (2, 1)]

Завершённый путь: [0, 2, 1, 3, 4, 0], Итоговая стоимость: 37

Найден новый лучший путь!

Ветка 2: Исключаем ребро 2 -> 1

=============================

Текущий путь: [(3, 4), (1, 3), (0, 2)], Текущий город: 2, Текущая нижняя оценка: 37.0

Редукция матрицы:

Матрица до редукции:

[inf, inf, inf, inf, inf]

[inf, inf, inf, inf, inf]

[inf, inf, inf, inf, inf]

[inf, inf, inf, inf, inf]

[0, inf, inf, inf, inf]

Из строки 0 ничего не вычли

Из строки 1 ничего не вычли

Из строки 2 ничего не вычли

Из строки 3 ничего не вычли

Из строки 4 ничего не вычли

Из столбца 0 ничего не вычли

Из столбца 1 ничего не вычли

Из столбца 2 ничего не вычли

Из столбца 3 ничего не вычли

Из столбца 4 ничего не вычли

Матрица после редукции:

[inf, inf, inf, inf, inf]

[inf, inf, inf, inf, inf]

[inf, inf, inf, inf, inf]

[inf, inf, inf, inf, inf]

[0, inf, inf, inf, inf]

Общее сокращение: 0.0

Нижняя граница стоимости: 37.0, Текущая лучшая стоимость: 37

Обрезка ветви (граница выше текущего лучшего решения)

Ветка 2: Исключаем ребро 0 -> 2

=============================

Текущий путь: [(3, 4), (1, 3)], Текущий город: 3, Текущая нижняя оценка: 37.0

Редукция матрицы:

Матрица до редукции:

[inf, 11, inf, inf, inf]

[inf, inf, inf, inf, inf]

[0, 0, inf, inf, inf]

[inf, inf, inf, inf, inf]

[0, inf, 3, inf, inf]

Из строки 0 вычли 11

Из строки 1 ничего не вычли

Из строки 2 ничего не вычли

Из строки 3 ничего не вычли

Из строки 4 ничего не вычли

Из столбца 0 ничего не вычли

Из столбца 1 ничего не вычли

Из столбца 2 вычли 3

Из столбца 3 ничего не вычли

Из столбца 4 ничего не вычли

Матрица после редукции:

[inf, 0, inf, inf, inf]

[inf, inf, inf, inf, inf]

[0, 0, inf, inf, inf]

[inf, inf, inf, inf, inf]

[0, inf, 0, inf, inf]

Общее сокращение: 14.0

Нижняя граница стоимости: 51.0, Текущая лучшая стоимость: 37

Обрезка ветви (граница выше текущего лучшего решения)

Ветка 2: Исключаем ребро 1 -> 3

=============================

Текущий путь: [(3, 4)], Текущий город: 4, Текущая нижняя оценка: 33.0

Редукция матрицы:

Матрица до редукции:

[inf, 15, 0, 4, inf]

[16, inf, 5, inf, inf]

[0, 4, inf, 14, inf]

[inf, inf, inf, inf, inf]

[0, 0, 3, inf, inf]

Из строки 0 ничего не вычли

Из строки 1 вычли 5

Из строки 2 ничего не вычли

Из строки 3 ничего не вычли

Из строки 4 ничего не вычли

Из столбца 0 ничего не вычли

Из столбца 1 ничего не вычли

Из столбца 2 ничего не вычли

Из столбца 3 вычли 4

Из столбца 4 ничего не вычли

Матрица после редукции:

[inf, 15, 0, 0, inf]

[11, inf, 0, inf, inf]

[0, 4, inf, 10, inf]

[inf, inf, inf, inf, inf]

[0, 0, 3, inf, inf]

Общее сокращение: 9.0

Нижняя граница стоимости: 42.0, Текущая лучшая стоимость: 37

Обрезка ветви (граница выше текущего лучшего решения)

Ветка 2: Исключаем ребро 3 -> 4

=============================

Текущий путь: [], Текущий город: 0, Текущая нижняя оценка: 20.0

Редукция матрицы:

Матрица до редукции:

[inf, 18, 0, 4, 12]

[16, inf, 5, 0, 13]

[0, 7, inf, 14, 15]

[2, 0, 12, inf, inf]

[10, 13, 13, 0, inf]

Из строки 0 ничего не вычли

Из строки 1 ничего не вычли

Из строки 2 ничего не вычли

Из строки 3 ничего не вычли

Из строки 4 ничего не вычли

Из столбца 0 ничего не вычли

Из столбца 1 ничего не вычли

Из столбца 2 ничего не вычли

Из столбца 3 ничего не вычли

Из столбца 4 вычли 12

Матрица после редукции:

[inf, 18, 0, 4, 0]

[16, inf, 5, 0, 1]

[0, 7, inf, 14, 3]

[2, 0, 12, inf, inf]

[10, 13, 13, 0, inf]

Общее сокращение: 12.0

Нижняя граница стоимости: 32.0, Текущая лучшая стоимость: 37

Для элемента [0][2] штраф 5

Для элемента [0][4] штраф 1

Для элемента [1][3] штраф 1

Для элемента [2][0] штраф 5

Для элемента [3][1] штраф 9

Для элемента [4][3] штраф 10

Выбрано ребро 4 -> 3 с максимальным коэффициентом 10

Ветка 1: Включаем ребро 4 -> 3, его вес: 6

Исключаем ребро 3 -> 4

=============================

Текущий путь: [(4, 3)], Текущий город: 3, Текущая нижняя оценка: 32.0

Редукция матрицы:

Матрица до редукции:

[inf, 18, 0, inf, 0]

[16, inf, 5, inf, 1]

[0, 7, inf, inf, 3]

[2, 0, 12, inf, inf]

[inf, inf, inf, inf, inf]

Из строки 0 ничего не вычли

Из строки 1 вычли 1

Из строки 2 ничего не вычли

Из строки 3 ничего не вычли

Из строки 4 ничего не вычли

Из столбца 0 ничего не вычли

Из столбца 1 ничего не вычли

Из столбца 2 ничего не вычли

Из столбца 3 ничего не вычли

Из столбца 4 ничего не вычли

Матрица после редукции:

[inf, 18, 0, inf, 0]

[15, inf, 4, inf, 0]

[0, 7, inf, inf, 3]

[2, 0, 12, inf, inf]

[inf, inf, inf, inf, inf]

Общее сокращение: 1.0

Нижняя граница стоимости: 33.0, Текущая лучшая стоимость: 37

Для элемента [0][2] штраф 4

Для элемента [0][4] штраф 0

Для элемента [1][4] штраф 4

Для элемента [2][0] штраф 5

Для элемента [3][1] штраф 9

Выбрано ребро 3 -> 1 с максимальным коэффициентом 9

Ветка 1: Включаем ребро 3 -> 1, его вес: 4

Исключаем ребро 1 -> 4

=============================

Текущий путь: [(4, 3), (3, 1)], Текущий город: 1, Текущая нижняя оценка: 33.0

Редукция матрицы:

Матрица до редукции:

[inf, inf, 0, inf, 0]

[15, inf, 4, inf, inf]

[0, inf, inf, inf, 3]

[inf, inf, inf, inf, inf]

[inf, inf, inf, inf, inf]

Из строки 0 ничего не вычли

Из строки 1 вычли 4

Из строки 2 ничего не вычли

Из строки 3 ничего не вычли

Из строки 4 ничего не вычли

Из столбца 0 ничего не вычли

Из столбца 1 ничего не вычли

Из столбца 2 ничего не вычли

Из столбца 3 ничего не вычли

Из столбца 4 ничего не вычли

Матрица после редукции:

[inf, inf, 0, inf, 0]

[11, inf, 0, inf, inf]

[0, inf, inf, inf, 3]

[inf, inf, inf, inf, inf]

[inf, inf, inf, inf, inf]

Общее сокращение: 4.0

Нижняя граница стоимости: 37.0, Текущая лучшая стоимость: 37

Обрезка ветви (граница выше текущего лучшего решения)

Ветка 2: Исключаем ребро 3 -> 1

=============================

Текущий путь: [(4, 3)], Текущий город: 3, Текущая нижняя оценка: 33.0

Редукция матрицы:

Матрица до редукции:

[inf, 18, 0, inf, 0]

[15, inf, 4, inf, 0]

[0, 7, inf, inf, 3]

[2, inf, 12, inf, inf]

[inf, inf, inf, inf, inf]

Из строки 0 ничего не вычли

Из строки 1 ничего не вычли

Из строки 2 ничего не вычли

Из строки 3 вычли 2

Из строки 4 ничего не вычли

Из столбца 0 ничего не вычли

Из столбца 1 вычли 7

Из столбца 2 ничего не вычли

Из столбца 3 ничего не вычли

Из столбца 4 ничего не вычли

Матрица после редукции:

[inf, 11, 0, inf, 0]

[15, inf, 4, inf, 0]

[0, 0, inf, inf, 3]

[0, inf, 10, inf, inf]

[inf, inf, inf, inf, inf]

Общее сокращение: 9.0

Нижняя граница стоимости: 42.0, Текущая лучшая стоимость: 37

Обрезка ветви (граница выше текущего лучшего решения)

Ветка 2: Исключаем ребро 4 -> 3

=============================

Текущий путь: [], Текущий город: 0, Текущая нижняя оценка: 32.0

Редукция матрицы:

Матрица до редукции:

[inf, 18, 0, 4, 0]

[16, inf, 5, 0, 1]

[0, 7, inf, 14, 3]

[2, 0, 12, inf, inf]

[10, 13, 13, inf, inf]

Из строки 0 ничего не вычли

Из строки 1 ничего не вычли

Из строки 2 ничего не вычли

Из строки 3 ничего не вычли

Из строки 4 вычли 10

Из столбца 0 ничего не вычли

Из столбца 1 ничего не вычли

Из столбца 2 ничего не вычли

Из столбца 3 ничего не вычли

Из столбца 4 ничего не вычли

Матрица после редукции:

[inf, 18, 0, 4, 0]

[16, inf, 5, 0, 1]

[0, 7, inf, 14, 3]

[2, 0, 12, inf, inf]

[0, 3, 3, inf, inf]

Общее сокращение: 10.0

Нижняя граница стоимости: 42.0, Текущая лучшая стоимость: 37

Обрезка ветви (граница выше текущего лучшего решения)

Оптимальный путь: [0, 2, 1, 3, 4, 0]

Общая стоимость: 37

2-приближённое решение ADO MOD 2:

Минимальное остовное дерево (MST):

Добавлено ребро 0 - 2 с весом 2

Добавлено ребро 1 - 3 с весом 4

Добавлено ребро 0 - 3 с весом 6

Добавлено ребро 3 - 4 с весом 6

Дерево из MST {0: [2, 3], 1: [3], 2: [0], 3: [1, 0, 4], 4: [3]}

DFS обход дерева

Добавили вершину 0 в путь

Добавили вершину 2 в путь

Добавили вершину 3 в путь

Добавили вершину 1 в путь

Добавили вершину 4 в путь

Приближённый маршрут: [0, 2, 3, 1, 4, 0], Стоимость: 57

Маршрут: [0, 2, 3, 1, 4, 0]

Приблизительная стоимость: 57

**Выводы.**

Был реализован алгоритм, решающий задачу с использованием приближённого алгоритма АДО МОД и оптимального алгоритма Литтла.

**Приложение А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Имя файла: littleBnbWithComments.cpp

#include <limits>

#include <vector>

#include <chrono>

#include <random>

#include <fstream>

#include <sstream>

#include <iostream>

#include <algorithm>

#include <unordered\_map>

#include <set>

#include <iomanip>

#include <unordered\_set>

#include <map>

const int INF = std::numeric\_limits<int>::max();

class Matrix {

public:

std::vector<std::vector<int>> \_matrix; // Двумерный вектор для хранения матрицы

int \_size; // Размер матрицы (количество строк и столбцов)

public:

// Конструктор, инициализирующий матрицу нулями заданного размера

Matrix(int size) {

\_size = size;

\_matrix = std::vector<std::vector<int>>(\_size, std::vector<int>(\_size, 0));

}

// Конструктор, инициализирующий матрицу заданными данными

Matrix(std::vector<std::vector<int>> matrix) {

\_size = matrix.size();

\_matrix = matrix;

}

Matrix() {};

// Устанавливает значение val в ячейку (row, col)

void setCell(int row, int col, int val) {

\_matrix[row][col] = val;

}

// Возвращает значение в ячейке (row, col)

int getCell(int row, int col) {

return \_matrix[row][col];

}

// Возвращает размер матрицы

int getSize() {

return \_size;

}

// Удаляет строку и столбец, заменяя их значениями INF

void deleteRowCol(int row, int col) {

for (int i = 0; i < \_size; i++) {

for (int j = 0; j < \_size; j++) {

if (i == row || j == col) {

setCell(i, j, INF);

}

}

}

}

// Возвращает минимальное значение в строке row

int getMinInRow(int row) {

int min = INF;

for (int j = 0; j < \_size; j++) {

int curr = \_matrix[row][j];

if (curr < min) {

min = curr;

}

}

return min;

}

// Возвращает минимальное значение в столбце col

int getMinInCol(int col) {

int min = INF;

for (int i = 0; i < \_size; i++) {

int curr = \_matrix[i][col];

if (curr < min) {

min = curr;

}

}

return min;

}

// Возвращает всю матрицу в виде вектора

std::vector<std::vector<int>> getMatrix(){

return \_matrix;

}

// Выводит матрицу в консоль

void print() {

std::cout << std::string(50, '-') << '\n';

for (int i = 0; i < \_size; i++) {

for (int j = 0; j < \_size; j++) {

// Если значение INF, выводим -1

\_matrix[i][j] == INF ? std::cout << "-1" : std::cout << \_matrix[i][j];

j == \_size - 1 ? std::cout << '\n' : std::cout << ' ';

}

}

std::cout << std::string(50, '-') << '\n';

}

// Генерирует случайную матрицу, может быть симметричной

void generate(bool is\_symmetric = false, int min = 1, int max = 100) {

unsigned seed = std::chrono::steady\_clock::now().time\_since\_epoch().count();

std::mt19937 gen(seed); // Генератор случайных чисел

std::uniform\_int\_distribution<> dist(min, max); // Распределение чисел в заданном диапазоне

for (int i = 0; i < \_size; i++) {

for (int j = 0; j < \_size; j++) {

if (i == j) {

\_matrix[i][j] = INF; // Заполняем диагональ значением INF

} else {

if (is\_symmetric) {

if (i < j) {

\_matrix[i][j] = dist(gen); // Заполняем только верхний треугольник

} else {

\_matrix[i][j] = \_matrix[j][i]; // Делаем матрицу симметричной

}

} else {

\_matrix[i][j] = dist(gen); // Заполняем случайными числами

}

}

}

}

}

//сохранить в файл

void saveToFile(const std::string& filename) {

std::ofstream file(filename);

if (file.is\_open()) {

for (int i = 0; i < \_size; i++) {

for (int j = 0; j < \_size; j++) {

file << \_matrix[i][j] << " ";

}

file << '\n';

}

}

file.close();

}

//загрузить матрицу из файла

void loadFromFile(const std::string& filename) {

std::ifstream file(filename);

std::vector<int> row;

\_matrix.clear();

int val;

std::string line;

while (std::getline(file, line)) {

std::istringstream iss(line);

row.clear();

while (iss >> val) {

row.push\_back(val);

}

\_matrix.push\_back(row);

}

\_size = row.size();

}

void readMatrixFromConsole() {

int n;

std::cin >> n;

std::vector<std::vector<int>> data(n, std::vector<int>(n));

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

std::cin >> data[i][j];

if (i == j){

data[i][j] = INF;

}

}

}

\_matrix = data;

\_size = data.size();

}

};

class BranchAndBound {

public:

int reduce(Matrix& matrix) {

int bound = 0;

int n = matrix.getSize();

std::cout << "\nРедукция матрицы:" << std::endl;

std::cout << "Матрица до редукции:" << std::endl;

matrix.print();

// Вычитание минимального значения из каждой строки

for (int i = 0; i < n; i++) {

int min = matrix.getMinInRow(i);

if (min > 0 && min != INF) {

for (int j = 0; j < n; j++) {

if (matrix.getCell(i, j) != INF) {

matrix.setCell(i, j, matrix.getCell(i, j) - min);

}

}

std::cout << "Из строки " << i << " вычли " << min << std::endl;

bound += min;

} else {

std::cout << "Из строки " << i << " ничего не вычли" << std::endl;

}

}

// Вычитание минимального значения из каждого столбца

for (int j = 0; j < n; j++) {

int min = matrix.getMinInCol(j);

if (min > 0 && min != INF) {

for (int i = 0; i < n; i++) {

if (matrix.getCell(i, j) != INF) {

matrix.setCell(i, j, matrix.getCell(i, j) - min);

}

}

std::cout << "Из столбца " << j << " вычли " << min << std::endl;

bound += min;

} else {

std::cout << "Из столбца " << j << " ничего не вычли" << std::endl;

}

}

std::cout << "Матрица после редукции:" << std::endl;

matrix.print();

std::cout << "Общее сокращение: " << bound << "\n" << std::endl;

return bound;

}

std::pair<int, int> calcBestArc(Matrix& matrix) {

std::vector<std::pair<int, int>> coords;

std::cout << "\nПоиск лучшей дуги (ребра):" << std::endl;

// Находим все нулевые элементы в матрице (кандидаты на выбор ребра)

for (int i = 0; i < matrix.getSize(); i++) {

for (int j = 0; j < matrix.getSize(); j++) {

if (matrix.getCell(i, j) == 0) {

coords.push\_back({i, j});

}

}

}

std::cout << "Найденные нулевые элементы (кандидаты на ребро):" << std::endl;

for (auto& coord : coords) {

std::cout << "(" << coord.first << ", " << coord.second << ") ";

}

std::cout << std::endl;

int maxdelta = -1;

std::pair<int, int> bestCoord = {-1, -1};

// Рассчитываем "цену" отказа от каждого нулевого элемента и выбираем наилучший

for (auto &coord: coords) {

matrix.setCell(coord.first, coord.second, INF);

int minInRow = matrix.getMinInRow(coord.first);

int minInCol = matrix.getMinInCol(coord.second);

if (minInRow == INF) {

minInRow = 0;

}

if (minInCol == INF) {

minInCol = 0;

}

int delta = minInRow + minInCol;

matrix.setCell(coord.first, coord.second, 0);

std::cout << "Анализируем ноль в (" << coord.first << ", " << coord.second << "):" << std::endl;

std::cout << "Минимум в строке: " << minInRow << ", минимум в столбце: " << minInCol << ", дельта: " << delta << std::endl;

if (maxdelta < delta) {

maxdelta = delta;

bestCoord = {coord.first, coord.second};

}

}

std::cout << "Выбрано ребро: (" << bestCoord.first << ", " << bestCoord.second << ") с дельтой " << maxdelta << "\n";

return bestCoord;

}

std::pair<int, int> calcForbiddenArc(Matrix& matrix, std::vector<std::pair<int, int>>& path, int from, int to) {

int X = from, Y = to;

std::cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

std::cout << "Запрет дуги:";

// 1. Найти все рёбра, которые ведут к X

bool cond = true;

int incoming\_to = X;

while (cond) {

cond = false;

for (const auto& [u, v] : path) {

if (v == incoming\_to) { // Если из вершины v есть ребро к X

cond = true;

incoming\_to = u; // Вершина, откуда приходит ребро к X (W → X)

break;

}

}

}

// 2. Найти все рёбра, которые выходят из Y

cond = true;

int outgoing\_from = Y;

while (cond) {

cond = false;

for (const auto& [u, v] : path) {

if (u == outgoing\_from) { // Если из вершины Y выходит ребро к v

cond = true;

outgoing\_from = v; // Вершина, в которую ведёт ребро от Y (Y → Z)

break;

}

}

}

// Если есть вход в X (W → X) и выход из Y (Y → Z), то запрещаем ребро Z → W

if (incoming\_to != X || outgoing\_from != Y) {

std::cout << outgoing\_from << " " << incoming\_to << "\n";

return {outgoing\_from, incoming\_to};

}

// Если других связей нет, то просто запрещаем Y → X

std::cout << X << " " << Y << "\n";

return {Y, X};

}

void calcBnB(Matrix& matrix, int cost, std::vector<std::pair<int, int>>& path, std::vector<std::pair<int, int>>& bestPath, int& bestCost) {

std::cout << "\n=============================" << std::endl;

std::cout << "Текущий путь: ";

for (auto& p : path) {

std::cout << "(" << p.first << " -> " << p.second << ") ";

}

std::cout << "\nТекущая стоимость: " << cost << ", Лучшая известная стоимость: " << bestCost << std::endl;

// Базовый случай: если путь содержит (n - 1) рёбер, замыкаем цикл

if (path.size() == matrix.\_size - 1) {

if (cost < bestCost) {

bestCost = cost;

bestPath = path;

std::cout << "Найден новый лучший путь с стоимостью " << bestCost << ":" << std::endl;

for (auto& p : bestPath) {

std::cout << "(" << p.first << " -> " << p.second << ") ";

}

std::cout << std::endl;

for (int i = 0; i < matrix.getSize(); i++) {

for (int j = 0; j < matrix.getSize(); j++) {

if (matrix.getCell(i, j) != INF) {

bestPath.push\_back({i, j});

}

}

}

}

return;

}

// Проверяем, не превысила ли текущая стоимость наилучшее найденное решение

if (cost >= bestCost) {

std::cout << "Обрезка ветви: текущая стоимость " << cost << " >= " << bestCost << std::endl;

return;

}

// Шаг 1: Редукция матрицы и обновление стоимости

int reduction = reduce(matrix);

int totalLowerBound = cost + reduction;

std::cout << "После редукции: новая стоимость " << totalLowerBound << " (редукция на " << reduction << ")" << std::endl;

if (totalLowerBound >= bestCost) {

std::cout << "Обрезка ветви: нижняя граница " << totalLowerBound << " >= " << bestCost << std::endl;

return;

}

// Шаг 2: Выбор наилучшего ребра

std::pair<int, int> arc = calcBestArc(matrix);

int i = arc.first, j = arc.second;

if (i != -1) {

std::cout << "Выбрано ребро (" << i << " -> " << j << ")" << std::endl;

// Ветка 1: Включаем ребро (i -> j)

Matrix newMatrix1 = Matrix(matrix.\_matrix);

std::pair<int, int> forb\_arc = calcForbiddenArc(newMatrix1, path, i, j);

path.push\_back(arc);

std::cout << "Добавили ребро (" << i << " -> " << j << ") в путь" << std::endl;

newMatrix1.setCell(forb\_arc.first, forb\_arc.second, INF);

std::cout << "Запрещено обратное ребро (" << forb\_arc.first << " -> " << forb\_arc.second << ")" << std::endl;

newMatrix1.deleteRowCol(i, j);

std::cout << "Удалены строка " << i << " и столбец " << j << " из матрицы" << std::endl;

std::cout << "Переход в рекурсивный вызов (ветка 1: включение ребра)" << std::endl;

calcBnB(newMatrix1, totalLowerBound, path, bestPath, bestCost);

// Ветка 2: Исключаем ребро (i -> j)

path.pop\_back();

std::cout << "Возвращаемся назад, убираем ребро (" << i << " -> " << j << ") из пути" << std::endl;

matrix.setCell(i, j, INF);

std::cout << "Запрещаем ребро (" << i << " -> " << j << "), переход к альтернативной ветви" << std::endl;

calcBnB(matrix, totalLowerBound, path, bestPath, bestCost);

}

}

// Восстановление пути из списка рёбер

std::vector<int> formWay(std::vector<std::pair<int, int>> &arcs, int start\_vertex) {

std::vector<int> resR, resL;

int vertex = start\_vertex;

resR.push\_back(vertex);

// Двигаемся вперёд по рёбрам

bool cond = true;

while (cond) {

cond = false;

for (const auto &[u, v] : arcs) {

if (u == vertex) {

cond = true;

vertex = v;

resR.push\_back(vertex);

break;

}

}

}

// Двигаемся назад по рёбрам

vertex = start\_vertex;

cond = true;

while (cond) {

cond = false;

for (const auto &[u, v] : arcs) {

if (v == vertex) {

cond = true;

vertex = u;

resL.insert(resL.begin(), vertex);

break;

}

}

}

// Объединяем левую и правую части пути

std::vector<int> res;

res.insert(res.end(), resR.begin(), resR.end());

res.insert(res.end(), resL.begin(), resL.end());

return res;

}

};

int main() {

Matrix matrix;

//matrix.readMatrixFromConsole();

matrix.loadFromFile("m5.txt");

matrix.print();

std::vector<std::pair<int, int>> path = {};

std::vector<std::pair<int, int>> bestPath = {};

int bestCost = INF;

BranchAndBound desicion;

desicion.calcBnB(matrix, 0, path, bestPath, bestCost);

std::vector<int> res = desicion.formWay(bestPath, 0);

std::cout << "\n Best Path: ";

for (int i = 0; i < res.size() - 1; i++) {

std::cout << res[i] << " ";

}

std::cout << res[res.size() - 1];

std::cout << "\n Cost: ";

std::cout << std::fixed << std::setprecision(1) << (float)bestCost << "\n";

}

Имя файла: modTspWithComments.cpp

#include <limits>

#include <vector>

#include <chrono>

#include <random>

#include <fstream>

#include <sstream>

#include <iostream>

#include <queue>

#include <algorithm>

#include <unordered\_map>

#include <set>

#include <iomanip>

#include <unordered\_set>

#include <iostream>

#include <vector>

#include <climits>

#include <stack>

#include <algorithm>

#include <map>

const int INF = std::numeric\_limits<int>::max(); // Константа, обозначающая бесконечность (используется для исключения путей)

class Matrix {

private:

    std::vector<std::vector<double>> \_matrix; // Двумерный вектор для хранения матрицы стоимостей

    int \_size;                             // Размер матрицы (число городов)

public:

    // Конструктор для создания квадратной матрицы заданного размера

    Matrix(int size) {

        \_size = size;

        \_matrix = std::vector<std::vector<double>>(\_size, std::vector<double>(\_size, 0));

    }

    // Конструктор для инициализации матрицы готовыми значениями

    Matrix(std::vector<std::vector<double>> matrix) {

        \_size = matrix.size();

        \_matrix = matrix;

    }

    Matrix() {};

    // Установка значения в ячейку матрицы

    void setCell(int row, int col, int val) {

        \_matrix[row][col] = val;

    }

    // Получение значения из ячейки матрицы

    double getCell(int row, int col) {

        return \_matrix[row][col];

    }

    // Получение размера матрицы

    int getSize() {

        return \_size;

    }

    // Удаление строки и столбца из матрицы (заменяя их на INF, чтобы они не учитывались)

    void deleteRowCol(int row, int col) {

        for (int i = 0; i < \_size; i++)

        {

            for (int j = 0; j < \_size; j++)

            {

                if (i == row || j == col)

                {

                    setCell(i, j, INF);

                }

            }

        }

    }

    // Нахождение минимального элемента в строке

    int getMinInRow(int row) {

        int min = INF;

        for (int j = 0; j < \_size; j++)

        {

            int curr = \_matrix[row][j];

            if (curr < min)

            {

                min = curr;

            }

        }

        return min;

    }

    // Нахождение минимального элемента в столбце

    int getMinInCol(int col) {

        int min = INF;

        for (int i = 0; i < \_size; i++)

        {

            int curr = \_matrix[i][col];

            if (curr < min)

            {

                min = curr;

            }

        }

        return min;

    }

    // Вывод матрицы в консоль для отладки

    void print() {

        std::cout << std::string(50, '-') << '\n';

        for (int i = 0; i < \_size; i++)

        {

            for (int j = 0; j < \_size; j++)

            {

                \_matrix[i][j] == INF ? std::cout << "-1" : std::cout << \_matrix[i][j];

                j == \_size - 1 ? std::cout << '\n' : std::cout << ' ';

            }

        }

        std::cout << std::string(50, '-') << '\n';

    }

    // Генерация матрицы

    void generate(bool is\_symmetric = false, int min = 1, int max = 100) {

        unsigned seed = std::chrono::steady\_clock::now().time\_since\_epoch().count();

        std::mt19937 gen(seed);

        std::uniform\_int\_distribution<> dist(min, max);

        for (int i = 0; i < \_size; i++) {

            for (int j = 0; j < \_size; j++) {

                if (i == j) {

                    \_matrix[i][j] = INF;

                }

                else {

                    if (is\_symmetric) {

                        if (i < j) {

                            \_matrix[i][j] = dist(gen);

                        }

                        else {

                            \_matrix[i][j] = \_matrix[j][i];

                        }

                    }

                    else {

                        \_matrix[i][j] = dist(gen);

                    }

                }

            }

        }

    }

    void readMatrixFromConsole() {

        std::vector<std::vector<double>> data; // Матрица весов

        std::string line;

        int n = 0; // Определяем размерность матрицы

        // Пропускаем символ новой строки после числа

        std::cin.ignore();

        // Читаем строки, пока есть данные

        while (std::getline(std::cin, line)) {

            std::stringstream ss(line);

            std::vector<double> row;

            double value;

            while (ss >> value) {

                row.push\_back(value);

            }

            if (row.empty()) break; // Если пустая строка, завершаем чтение

            data.push\_back(row);

        }

        n = data.size(); // Определяем размерность матрицы

        // Заменяем -1 на INF (обозначение отсутствия пути)

        for (int i = 0; i < n; ++i) {

            for (int j = 0; j < n; ++j) {

                if (data[i][j] == -1) {

                    data[i][j] = INF;

                }

            }

        }

        \_matrix = data;

        \_size = data.size();

    }

    //сохранить в файл

    void saveToFile(const std::string& filename) const {

        std::ofstream file(filename);

        if (file.is\_open()) {

            for (size\_t i = 0; i < \_size; i++) {

                for (size\_t j = 0; j < \_size; j++) {

                    file << \_matrix[i][j] << " ";

                }

                file << '\n';

            }

        }

        file.close();

    }

    //считать с файла

    void loadFromFile(const std::string& filename) {

        std::ifstream file(filename);

        std::vector<double> row;

        \_matrix.clear();

        std::string line;

        double val;

        while (std::getline(file, line)) {

            std::istringstream iss(line);

            row.clear();

            while (iss >> val) {

                row.push\_back(val);

            }

            \_matrix.push\_back(row);

        }

        \_size = \_matrix.size();

    }

};

class ModTSP {

public:

std::vector<std::pair<int, int>> primMST(Matrix& matrix) {

    int n = matrix.getSize();

    std::vector<double> minWeight(n, INF);  // Минимальные веса рёбер

    std::vector<int> parent(n, -1);  // Родитель каждой вершины в MST

    std::vector<bool> inMST(n, false);  // Включена ли вершина в MST

    minWeight[0] = 0;  // Начинаем с первой вершины

    std::priority\_queue<std::pair<double, int>, std::vector<std::pair<double, int>>, std::greater<>> pq;

    pq.push({0.0, 0});

    std::cout << "Построение минимального остовного дерева (MST) алгоритмом Прима:\n";

    while (!pq.empty()) {

        int u = pq.top().second;

        pq.pop();

        if (inMST[u]) continue;

        inMST[u] = true;

        std::cout << "Добавлена вершина " << u << " в MST.\n";

        // Проход по соседям вершины u

        for (int v = 0; v < n; ++v) {

            double weight = matrix.getCell(u, v);

            if (weight != INF && !inMST[v] && weight < minWeight[v]) {

                minWeight[v] = weight;

                parent[v] = u;

                pq.push({weight, v});

                std::cout << "Рассмотрено ребро (" << u << " -> " << v << "), вес: " << weight << "\n";

            }

        }

    }

    std::vector<std::pair<int, int>> mstEdges;

    std::cout << "Готовый список рёбер MST:\n";

    for (int v = 1; v < n; ++v) {

        if (parent[v] != -1) {

            mstEdges.push\_back({parent[v], v});

            std::cout << "(" << parent[v] << " -> " << v << ")\n";

        }

    }

    return mstEdges;

}

// Функция выполняет обход минимального остовного дерева (MST) в глубину (DFS)

// и возвращает путь обхода в виде списка вершин.

std::vector<int> dfsMST(const std::vector<std::vector<std::pair<int, double>>>& mstAdj, int start\_vertex) {

    std::vector<int> path;               // Хранит порядок посещения вершин

    std::vector<bool> visited(mstAdj.size(), false); // Вектор для отслеживания посещённых вершин

    std::vector<int> stack;              // Имитация стека для DFS (глубинного обхода)

    stack.push\_back(start\_vertex); // Начинаем обход с заданной вершины

    std::cout << "Обход MST в глубину (DFS), начиная с вершины " << start\_vertex << ":\n";

    // Пока стек не пуст, продолжаем обход

    while (!stack.empty()) {

        int node = stack.back(); // Берём верхний элемент стека

        stack.pop\_back();        // Удаляем его из стека

        if (visited[node]) continue; // Если вершина уже посещена, пропускаем её

        visited[node] = true;  // Отмечаем вершину как посещённую

        path.push\_back(node);  // Добавляем её в путь обхода

        std::cout << "Посещена вершина " << node << "\n";

        // Получаем список соседей текущей вершины

        std::vector<std::pair<int, double>> neighbors = mstAdj[node];

        // Сортируем соседей по весу рёбер (чтобы посещать в порядке увеличения веса)

        std::sort(neighbors.begin(), neighbors.end(), [](const std::pair<int, double>& a, const std::pair<int, double>& b) {

            return a.second < b.second;

        });

        // Добавляем в стек соседей, которых ещё не посещали

        for (const auto& [v, \_] : neighbors) {

            if (!visited[v]) {

                stack.push\_back(v);

                std::cout << "Добавлен в стек сосед " << v << "\n";

            }

        }

    }

    return path; // Возвращаем список вершин в порядке их посещения

}

// Функция находит приближённое решение задачи коммивояжёра (TSP)

// с помощью минимального остовного дерева (MST).

// Используется метод Прима для построения MST, затем выполняется DFS-обход.

std::pair<double, std::vector<int>> approximateTSP(Matrix& matrix, int start\_vertex) {

    int n = matrix.getSize(); // Получаем размер матрицы (количество вершин в графе)

    std::cout << "\n==== Запуск приближённого решения TSP через MST ====\n";

    // 1. Строим минимальное остовное дерево (MST) с помощью алгоритма Прима.

    // Возвращается список рёбер остовного дерева.

    std::vector<std::pair<int, int>> mstEdges = primMST(matrix);

    // 2. Создаём список смежности для MST

    std::vector<std::vector<std::pair<int, double>>> mstAdj(n);

    for (const auto& [u, v] : mstEdges) {

        double weight = matrix.getCell(u, v); // Получаем вес ребра

        mstAdj[u].push\_back({v, weight}); // Добавляем в список смежности

        mstAdj[v].push\_back({u, weight}); // Граф неориентированный, добавляем обратное ребро

    }

    // 3. Выполняем обход MST в глубину (DFS), начиная с указанной стартовой вершины.

    std::vector<int> path = dfsMST(mstAdj, start\_vertex);

    // 4. Замыкаем цикл, возвращаясь к стартовой вершине

    double total\_cost = 0;

    if (!path.empty()) {

        path.push\_back(start\_vertex); // Добавляем стартовую вершину в конец пути

        std::cout << "Замыкаем путь: возвращаемся к стартовой вершине " << start\_vertex << "\n";

    }

    // 5. Вычисляем суммарную стоимость маршрута

    std::cout << "Рассчитанный путь TSP:\n";

    for (size\_t i = 0; i < path.size() - 1; ++i) {

        int u = path[i], v = path[i + 1];  // Берём текущую вершину и следующую в пути

        double edge\_cost = matrix.getCell(u, v); // Получаем вес ребра между ними

        // Если отсутствует ребро между вершинами, путь невозможен

        if (edge\_cost == INF) {

            std::cout << "Ошибка: отсутствует ребро (" << u << " -> " << v << "), путь невозможен.\n";

            return {INF, {}}; // Возвращаем бесконечность и пустой путь

        }

        total\_cost += edge\_cost; // Прибавляем стоимость ребра к общей стоимости маршрута

        std::cout << "(" << u << " -> " << v << "), вес: " << edge\_cost << "\n";

    }

    std::cout << "Общая стоимость маршрута: " << total\_cost << "\n";

    return {total\_cost, path}; // Возвращаем суммарную стоимость маршрута и сам путь

}

};

int main() {

    int start\_vertex;

    std::cin >> start\_vertex; // Считываем стартовую вершину

    Matrix matrix;

    matrix.loadFromFile("2.txt");

    matrix.print();

    std::vector<std::pair<int, int>> path = {};

    std::vector<std::pair<int, int>> bestPath = {};

    int bestCost = INF;

    ModTSP modTsp;

    auto result = modTsp.approximateTSP(matrix, start\_vertex);

    if (result.first == INF) {

        std::cout << "No solution\n";

    } else {

        std::cout << "Cost: " << (double)result.first << "\nPath: ";

        for (int v : result.second) {

            std::cout << v << " ";

        }

    }

    return 0;

}