МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Построение и анализ алгоритмов»

Тема: Кратчайшие пути в графах: коммивояжёр

Вариант 2

Студент гр. 3388 _____ Потоцкий С.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Задание

Решить задачу Коммивояжёра 2 различными способами. Алгоритм Литтла с модификацией: после приведения матрицы, к нижней оценке веса решения добавляется нижняя оценка суммарного веса остатка пути на основе МОД. Приближённый алгоритм: АБС. Начинать АБС со стартовой вершины.

Описание алгоритма

Алгоритм Литтла (метод ветвей и границ)

Инициализация:

Редуцирует исходную матрицу стоимостей. Вычисляет начальную нижнюю границу. Создает начальное состояние и помещает его в очередь с приоритетами.

Основной цикл:

Извлекает из очереди состояние с наименьшей нижней границей.

Если состояние представляет полное решение (все города посещены), возвращает это решение.

Иначе:

Выбирает ребро для ветвления. Создает два новых состояния: одно с включенным ребром, другое с исключенным ребром. Вычисляет нижние границы для новых состояний. Помещает новые состояния в очередь. Если очередь пуста и решение не найдено, алгоритм завершается.

Общее описание

Этот код реализует два алгоритма для решения задачи коммивояжера (TSP): алгоритм ближайшего соседа (АБС) и алгоритм Литтла (метод ветвей и границ). Задача коммивояжера заключается в нахождении самого дешевого маршрута, проходящего через все заданные города ровно по одному разу и возвращающегося в исходный город.

Функции и их назначение

nearestNeighborAlgorithm(const vector<vector<float>>& costMatrix):

Реализует жадный алгоритм ближайшего соседа. Начиная с произвольного города, на каждом шаге выбирает ближайший непосещенный город. Возвращает вектор, представляющий порядок посещения городов.

reduceMatrix(vector<vector<float>>& matrix):

Выполняет редукцию матрицы стоимостей. Вычитает минимальное значение из каждой строки и каждого столбца матрицы. Редукция используется в алгоритме Литтла для оценки нижней границы стоимости решения. Возвращает суммарное значение редукции.

calculateMSTBound(const vector<vector<float>>& matrix):

Вычисляет оценку снизу для стоимости решения на основе минимального остовного дерева (MST). Используется в алгоритме Литтла. Возвращает вес MST.

findBranchingEdge(const vector<vector<float>>& matrix):

Находит ребро для ветвления в алгоритме Литтла. Выбирает ребро с нулевой стоимостью, исключение или включение которого в маршрут окажет наибольшее влияние на нижнюю границу. Возвращает пару индексов (строка, столбец), представляющих выбранное ребро.

dfs(const vector<vector<int>>& graph, int node, int end, vector<bool>& visited, vector<int>& path):

Реализует поиск в глубину (DFS) для поиска пути между двумя вершинами в графе. Используется для проверки наличия циклов при включении ребер в алгоритме Литтла.

findPath(const vector<pair<int, int>>& edges, int start, int end, vector<int>& path, int n):

Определяет, существует ли путь между двумя заданными вершинами, используя поиск в глубину. Используется для обнаружения циклов при добавлении ребер.

findEdgeToExclude(const vector<pair<int, int>>& included, int from, int to, int n):

Находит ребро, которое нужно исключить, чтобы избежать образования цикла при включении нового ребра.

littleAlgorithm(vector<vector<float>> costMatrix):

Реализует алгоритм Литтла (метод ветвей и границ) для решения задачи коммивояжера. Использует очередь с приоритетами для хранения состояний (подзадач). На каждой итерации выбирает состояние с наименьшей нижней границей. Ветвится, включая или исключая ребро из маршрута. Возвращает вектор пар индексов, представляющих ребра, включенные в оптимальный маршрут.

edgesToPath(const vector<pair<int, int>>& edges, int n):

Преобразует список ребер в путь (последовательность городов). Используется для представления решения, найденного алгоритмом Литтла, в виде пути.

calculateTotalCost(const vector<int>& path, const vector<vector<float>>& originalCostMatrix):

Вычисляет общую стоимость маршрута на основе матрицы стоимостей.

Структуры данных

State: Представляет состояние в алгоритме Литтла. Содержит:

costMatrix: Текущая матрица стоимостей.

included: Список включенных в маршрут ребер.

excluded: Список исключенных из маршрута ребер.

lowerBound: Нижняя граница стоимости решения для данного состояния.

Оценка сложности алгоритма (Литтла)**:**

Временная сложность: O(n^2·2^n)

Редукция матрицы: O(n^2) для каждой строки и столбца (по n элементов).

Поиск ячейки с максимальным штрафом: O(n^2) для проверки всех

ячеек и вычисления штрафов.

Построение МОД: O(n^2) для создания графа и O(nlogn) для сортировки ребер (в худшем случае O(n^2) из-за числа ребер).

Количество узлов: В худшем случае алгоритм исследует все возможные деревья, что дает O(2^n) узлов.

Однако отсечение по границам значительно сокращает количество исследуемых узлов в среднем случае, делая алгоритм эффективнее полного перебора O(n!).

Оценка сложности алгоритма (жадного):

Временная сложность: O(n^2)

n-1 итераций по вершинам, n проверок при поиске ближайшего соседа на каждой итерации.

Вывод

В ходе лабораторной работы было написано решение задачи коммивояжера двумя способами. Алгоритм Литтла, основанный на методе ветвей и границ, находит оптимальный путь за приемлемое время по сравнению с полным перебором. Жадный алгоритм и аналитически и экспериментально проигрывает первому алгоритму в точности, однако имеет значительное преимущество в скорости работы.