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: Задача коммивояжёра.

Студент гр. 3388	____________________	Дубровин Д.Н.
Преподаватель	____________________	Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы.

Реализовать алгоритм Литтла и АДО МОД для решения задачи коммивояжёра методом ветвей и границ и нахождения 2-приближения.

Задание.

Ветви и границы.

В волшебной стране Алгоритмии великий маг, Гамильтон, задумал невероятное путешествие, чтобы связать все города страны заклятием процветания. Для этого ему необходимо посетить каждый город ровно один раз, создавая тропу благополучия, и вернуться обратно в столицу, используя минимум своих чародейских сил. Вашей задачей является помощь в прокладывании маршрута с помощью древнего и могущественного алгоритма ветвей и границ.

Карта дорог Алгоритмии перед Гамильтоном представляет собой полный граф, где каждый город соединён магическими порталами с каждым другим. Стоимость использования портала из города в город занимает определённое количество маны, и Гамильтон стремится минимизировать общее потребление магической энергии для закрепления проклятия.

Входные данные:

Первая строка содержит одно целое число N — количество городов). Города нумеруются последовательными числами от 0 до $N-1$

Следующие N строк содержат по N чисел каждая, разделённых пробелами, формируя таким образом матрицу стоимостей M.Каждый элемент $M_{i,j}$ этой матрицы представляет собой затраты маны на перемещение из города i в город j.

Выходные данные:

Первая строка: Список из N целых чисел, разделённых пробелами, обозначающих оптимальный порядок городов в магическом маршруте Гамильтона. В начале идёт город 0, с которого начинается маршрут, затем последующие города до тех пор, пока все они не будут посещены.

Вторая строка: Число, указывающее на суммарное количество израсходованной маны для завершения пути.

2-приближение.

Разработайте программу, которая решает задачу коммивояжера при помощи 2-приближенного алгоритма. В данной постановке задачи нужно вернуться в исходную вершину после прохождения всех остальных вершин. При обходе остовного дерева (MST) необходимо идти по минимальному допустимому ребру из текущего. Каждая вершина в графе обозначается неотрицательным числом, начиная с 0, каждое ребро имеет неотрицательный вес. В графе нет рёбер из вершины в саму себя, в матрице весов на месте таких отсутствующих рёбер стоит значение -1.

Пример входных данных

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

В первой строке указывается начальная вершина.

Далее идёт матрица весов.

В качестве выходных данных необходимо представить длину пути, полученного при помощи алгоритма. Следующей строкой необходимо представить путь, в котором перечислены вершины, по которым необходимо пройти от начальной вершины. Для приведённых в примере входных данных ответом будет:

91.92

2 3 0 4 1 2

Выполнение работы.

Метод ветвей и границ:

get_lower_bound() - функция вычисления нижней границы стоимости пути:

- Для текущей вершины берет минимальное ребро к непосещенным вершинам
- Для всех непосещенных вершин берет два минимальных ребра (или одно, если второе отсутствует)
- Суммирует эти минимальные значения для оценки нижней границы

tsp_branch_and_bound() - основная функция алгоритма:

- Использует приоритетную очередь (min-heap) для хранения частичных решений
- Начинает с начальной вершины (0)
- На каждом шаге расширяет частичные пути, добавляя непосещенные вершины
- Использует нижнюю границу для отсечения заведомо плохих вариантов

2-приближение:

prim_mst(graph, n, logger)

Алгоритм: Реализация алгоритма Прима для построения MST.

Шаги выполнения:

Инициализация:

- mst: пустой список смежности для дерева
- visited: массив отметок о посещении вершин
- pq: приоритетная очередь (min-heap) с кортежами (вес, вершина, родитель)

Начинаем с вершины 0 (добавляем в очередь с весом 0)

Основной цикл:

- Извлекаем вершину с минимальным весом
- Если вершина уже посещена - пропускаем
- Добавляем ребро в MST (кроме начальной вершины)
- Для всех смежных непосещенных вершин добавляем рёбра в очередь

- Возвращает построенное MST в виде списка смежности

tsp_2_approx(graph, start, n, logger)

Алгоритм: 2-приближенное решение TSP через MST и DFS.

Шаги выполнения:

Строит MST с помощью prim_mst()

Выполняет обход DFS по MST:

- Начинает с заданной стартовой вершины
- Рекурсивно посещает все вершины
- Сохраняет порядок обхода в path
- Добавляет стартовую вершину в конец для замыкания цикла
- Вычисляет общую длину пути по матрице весов
- Возвращает кортеж (длина_пути, список_вершин)

Оценка сложности.

Метод ветвей и границ:

Временная сложность

Худший случай: O(n!), где n - количество городов

Основные операции:

- Расчет нижней границы: $O(n^2)$ (из-за поиска минимальных ребер для каждой вершины)
- Вставка/извлечение из приоритетной очереди: O(log k), где k - размер очереди
- В худшем случае очередь может содержать до O(n!) элементов

Пространственная сложность

Основные затраты памяти:

- Хранение матрицы расстояний: $O(n^2)$
- Приоритетная очередь: в худшем случае O(n!) (но на практике значительно меньше)

- Хранение посещенных вершин и путей: $O(n)$ для каждого элемента в очереди
- Общая оценка: $O(n^2 + k*n)$, где k - максимальный размер очереди

<u>*2-приближение:*</u>

Временная сложность

Сложность алгоритма Прима: $O(n^2 log(n))$ (n раз нужно пройтись по n рёбрам из выбранной вершины и добавить в кучу за $O(log(n))$)

Сложность обхода МОД: Каждая вершина посещается за $O(n)$ и выполняется сортировка по соседним рёбрам.
Соседние рёбра – подмассив всех рёбер. Но в МОД всего $(n-1)*2$ рёбер (граф двунаправленный). Сложность сортировки такого массива была бы $n*log(n)$ если бы граф оказался звездой. Но сложность сортировок подмассивов меньше, чем сложность сортировки самого массива, поэтому все сортировки в алгоритме выполняются не более чем за $O(n*log(n))$. Итоговая сложность: $O(n) + O(n*log(n)) = O(n*log(n))$

Таким образом, сложность алгоритма равна $O(n^2 log(n)) + O(n*log(n)) = O(n^2 log(n))$.

Пространственная сложность

- Хранение графа: $O(n^2)$ (матрица смежности)
- Хранение MST: $O(n)$ (список смежности для n вершин)
- Приоритетная очередь: $O(n)$
- Хранение пути: $O(n)$

Итоговая пространственная сложность: $O(n^2)$

Вывод.

Были разработаны и проанализированы алгоритмы для решения задачи коммивояжёра. В качестве точного решения – алгоритм Литтла, в качестве 2-приближения - АДО МОД.