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: Коммивояджер

Вариант: 3

Студент гр. 3388 _____ Потапов Р.Ю.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы:

Изучить теоретические основы задачи коммивояджера. Решить ее с помощью МВиГ.

Задание:

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*N* (N*N* — количество городов). Города нумеруются последовательными числами от 0 до N−1*N*−1. Следующие N*N* строк содержат по N*N* чисел каждая, разделённых пробелами, формируя таким образом матрицу стоимостей M*M*. Каждый элемент Mi,j*Mi*,*j* этой матрицы представляет собой затраты маны на перемещение из города i*i* в город j*j*.

Выходные данные:

Первая строка: Список из N*N* целых чисел, разделённых пробелами,

обозначающих оптимальный порядок городов в магическом маршруте Гамильтона. В начале идёт город, 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.

Sample Input 1:

```
3
-1 1 3
3 -1 1
1 2 -1
```

Sample Output 1:

```
0 1 2
3.0
```

Sample Input 2:

```
4
-1 3 4 1
1 -1 3 4
9 2 -1 4
8 9 2 -1
```

Sample Output 2:

```
0 3 2 1
6.0
```

Выполнение работы

Описание алгоритма:

Алгоритм решения задачи коммивояжёра (TSP) методом ветвей и границ с двумя нижними оценками строится по принципу рекурсивного перебора и раннего отсечения (branch and bound). Он стремится найти оптимальный гамильтонов цикл минимальной стоимости, последовательно добавляя новые города к текущему маршруту и используя оценки, чтобы сократить пространство поиска. Ниже приводятся основные этапы работы алгоритма и краткое описание функций, задействованных в коде.

Алгоритм начинается с города 0, формируя частичное решение, и на каждом шаге добавляет одну ещё не посещённую вершину. При этом рассчитывается «нижняя граница» (bound), чтобы понять, возможно ли улучшить уже известное лучшее решение. Если bound оказывается не меньше текущего лучшего результата, ветвь поиска прекращается (отсечение), иначе алгоритм рекурсивно углубляется, пытаясь продолжить маршрут.

Основные этапы работы:

1. Инициализация. Алгоритм создаёт начальную матрицу расстояний и определяет лучшие найденные параметры (best_cost и best_route) как бесконечность, а текущий путь начинает из вершины 0. Дополнительно формируется min_edges — массив одношаговых минимальных рёбер, помогающий упорядочивать варианты перебора.

2. Расчёт нижних оценок. Используются две оценки: первая — полусумма двух минимальных рёбер для каждой вершины (two_min_edges_sum), вторая — вес минимального остовного дерева для оставшихся городов

(prim_mst) плюс два рёбра на вход и выход. Обе оценки вычисляются для ещё не посещённых вершин. Максимум из них даёт окончательную нижнюю границу bound. Если этот bound больше или равен уже найденному лучшему результату, дальнейший перебор бессмыслен и ветвь отсекается.

3. Рекурсивный перебор. На каждом уровне алгоритм выбирает очередную вершину (город), в которую можно перейти. Формируется новый путь, прибавляется стоимость дуги, и если результат всё ещё потенциально может улучшить best_cost, алгоритм углубляется дальше. Иначе ветвь отсекается.

4. Отсечения. Как только становится ясно, что текущее частичное решение уже не сможет улучшить имеющийся маршрут (или возникают непреодолимые ограничения, например отсутствие рёбер), ветвь завершается. Это ускоряет поиск и позволяет алгоритму обрабатывать значительно меньше вариантов.

5. Итоговый выбор. Когда все вершины посещены и последний город соединяется обратно с 0, сравнивается итоговая стоимость с best_cost. Если новый маршрут оказался короче, обновляются глобальные лучшая стоимость и лучший маршрут. После перебора всех потенциальных ветвей минимальный путь находится точно.

Описание функций:

1. solve_tsp_bb(n, dist). Основная функция, реализующая метод ветвей и границ. Инициализирует структуры для хранения лучших параметров, формирует вспомогательную информацию (min_edges), а затем вызывает рекурсивную процедуру branch_and_bound(0, {0}, [0], 0.0). По завершении возвращает найденный маршрут и его стоимость.

2. branch_and_bound(current, visited, path, cost). Рекурсивная функция перебора с отсечениями. Аргументы: текущий город, множество уже посещённых городов, текущая последовательность (path) и накопленная стоимость. Вычисляет нижнюю оценку через lower_bound, сравнивает с лучшей стоимостью и в случае, если bound оказывается меньше, генерирует кандидатов для продолжения маршрута и рекурсивно их обходит. Если все города посещены, проверяет возможность замкнуть цикл и при необходимости обновляет глобальный best_cost и best_route.

3. lower_bound(current_city, current_cost, visited). Функция вычисляет сумму уже накопленных затрат и максимум двух оценок: (а) полусуммы двух минимальных рёбер для каждой оставшейся вершины (two_min_edges_sum) и (б) MST для оставшихся (prim_mst) плюс два рёбра (из current_city к этим вершинам и обратно в 0). Результат играет роль bound, по которому производится отсечение.

4. two_min_edges_sum(n, dist, nodes). Считает полусумму двух самых коротких исходящих рёбер для каждой вершины в nodes. Если у какой-то вершины меньше двух допустимых рёбер, возвращается бесконечность. Это формирует первую нижнюю оценку для оставшихся городов.

5. prim_mst(n, dist, nodes). При помощи алгоритма Прима строит минимальное остовное дерево на заданном подмножестве вершин nodes и возвращает вес этого дерева. Используется в качестве второй нижней оценки, чтобы отражать минимум на «связку» непосещённых городов.

6. Генерация данных и вывод. В main или аналогичной функции код генерирует случайную матрицу dist, вызывает solve_tsp_bb для получения оптимального маршрута, а затем выводит итоговый путь, его стоимость и статистику (время, число отсечённых ветвей).

Оценка сложности алгоритма:

Временная сложность

Оценка сложности алгоритма сводится к тому, что метод ветвей и границ, рассматривая гамильтоновы циклы, в худшем случае может свестись к полному перебору всех перестановок городов, что даёт факториальную оценку сложности (O(n!)). Однако две нижние оценки (полусумма двух минимальных рёбер и MST+2 рёбра) позволяют эффективно «отсекать» многие ветви, существенно сокращая поиск на практике. Формально:

1. Худший случай (теоретический)

Без действенных отсечений ветви и границы могут просматривать все перестановки, то есть вплоть до O(n!) возможных маршрутов. Каждый шаг проверки для ветви включает расчёт bound, который сам по себе не требует слишком больших затрат (расположение вершин и вычисление MST обычно укладываются в полиномиальное время относительно n). Но общее число путей при крайне невыгодном случае всё равно даёт экспоненциальную по n (точнее факториальную) сложность.

2. Практическое ускорение за счёт оценок

В реальности две нижние оценки — «полусумма двух минимальных рёбер» и «MST + 2 рёбра» — быстро отбрасывают большую часть ветвей, поскольку, как только bound превысит уже найденный лучший результат, ветка не рассматривается дальше. Это снижает среднюю сложность алгоритма на большинстве «случайных» или метрически «гладких» входных данных. Зачастую на умеренных размерах (порядка 15–20 городов) метод ветвей и границ с такими отсечениями может работать достаточно быстро.

3. Память

Алгоритм хранит матрицу расстояний размера $n \times n$, а также структуры для промежуточных расчётов (например, массивы visited, min_edge и т. п.), что даёт $O(n^2)$ памяти. Плюс рекурсивный стек (глубина до n), и глобальные переменные для лучшего пути. Общая пространственная сложность остаётся полиномиальной относительно n.

Таким образом, формально метод ветвей и границ для TSP остаётся экспоненциальным ($O(n!)$), но благодаря эвристикам и двухуровневой оценке на практике достигается значительное сокращение перебираемых вариантов, что позволяет эффективно решать задачи среднего размера.

Визуализация

Для визуализации работы алгоритма была использована библиотека matplotlib

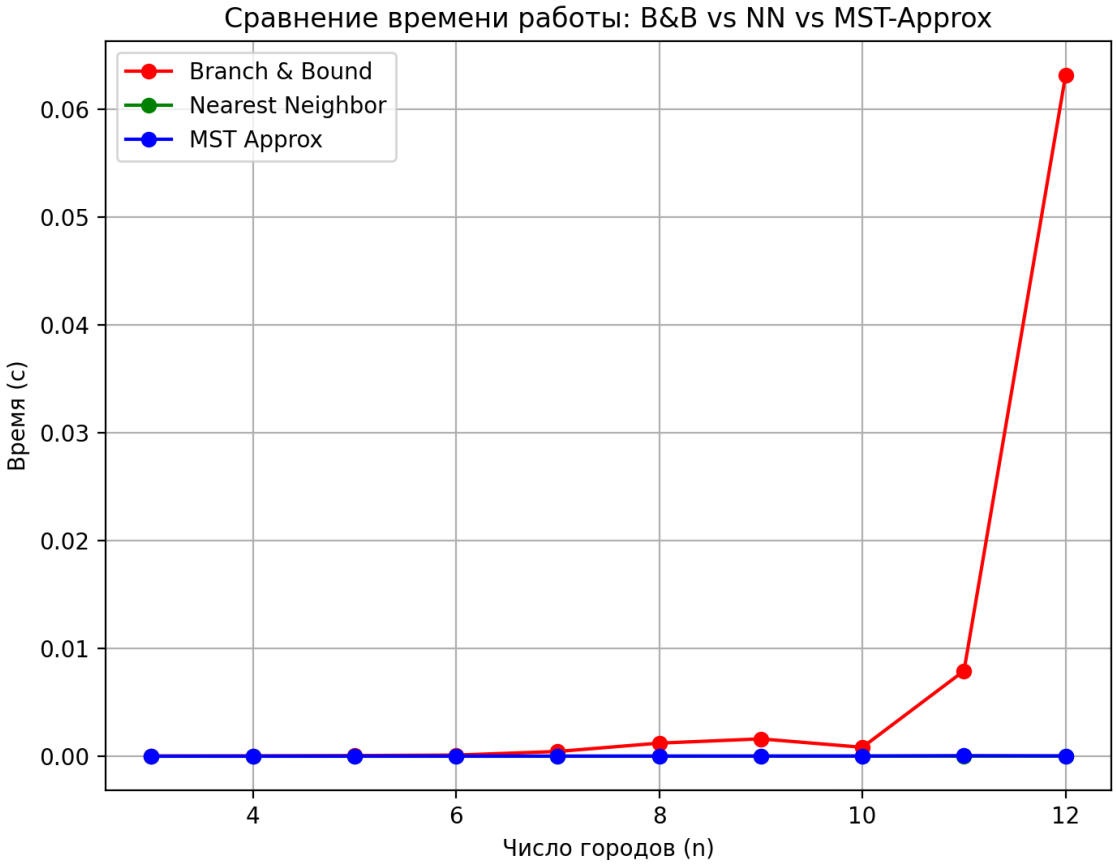


Рис. 1 Визуализация работы алгоритма.

Тестирование

Таблица 1. Тестирование.

Входные данные	*Выходные данные*
7	0 → 2 → 5 → 4 → 3 → 6 → 1 → 0 68.39
15	0 → 8 → 14 → 7 → 9 → 3 → 12 → 5 → 11 → 13 → 1 → 10 → 6 → 2 → 4 → 0 57.80
16	0 → 7 → 10 → 6 → 13 → 14 → 9 → 12 → 1 → 8 → 5 → 4 → 3 → 11 → 2 → 15 → 0 48.40
20	0 → 18 → 7 → 15 → 6 → 8 → 4 → 16 → 5 → 14 → 1 → 13 → 10 → 19 → 3 → 17 → 11 → 9 → 2 → 12 → 0 68.99

Исследование

В ходе данной лабораторной работы был рассмотрен алгоритм решения задачи коммивояжёра (TSP) методом ветвей и границ (МВиГ) с двумя разными нижними оценками, а также два эвристических алгоритма: «ближайший сосед» и «MST + обход в глубину (AMP)». Основная цель — найти маршрут (цикл), проходящий по всем городам ровно один раз и возвращающийся в исходную точку (город 0), при этом минимизируя суммарную стоимость пути.

2. Метод ветвей и границ (МВиГ) с двумя нижними оценками

Алгоритм ветвей и границ строит решение пошагово, рекурсивно добавляя города к текущему маршруту. При этом он вычисляет «нижнюю границу» (bound) возможной стоимости, чтобы отсеивать заведомо невыгодные ветви. Используются две оценки:

(а) Полусумма двух минимальных рёбер для каждой непосещённой вершины.

(б) Вес минимального остовного дерева (MST) на оставшихся вершинах плюс два «соединяющих» ребра (выход из текущего города и возвращение к 0).

На практике берётся максимум из (а) и (б), после чего прибавляется к уже накопленной стоимости. Если полученный bound не меньше лучшего найденного на текущий момент решения, ветвь больше не исследуется. Благодаря этому «прорезанию» число реально перебираемых вариантов существенно сокращается.

3. Функции в методе ветвей и границ

– solve_tsp_bb(n, dist). Главная функция, инициализирующая массивы для лучшего результата, а также вспомогательный список min_edges (хранит одно минимальное ребро из каждой вершины). Затем вызывает branch_and_bound(0, {0}, [0], 0.0), чтобы начать поиск из стартового города 0.

– branch_and_bound(current, visited, path, cost). Рекурсивная процедура: если все вершины уже посещены, проверяется возможность вернуться в 0 и при улучшении обновляется global best. Иначе вычисляется bound через lower_bound. Если bound превышает best_cost, ветвь отсекается. Иначе формируется набор «кандидатов» по эвристике dist[current][candidate] + min_edges[candidate], и для каждого вызывается рекурсия, увеличивая cost на вес дуги current→candidate.

– lower_bound(current_city, current_cost, visited). Считает текущую накопленную стоимость и прибавляет максимум из двух оценок: polusum двух самых коротких рёбер на оставшиеся вершины (two_min_edges_sum) и MST (prim_mst) плюс рёбра «вход-выход».

– two_min_edges_sum(n, dist, nodes). Для каждой вершины в nodes ищет два самых коротких исходящих рёбра, складывает и суммирует по всем вершинам, затем делит результат на 2. Если где-то меньше двух рёбер, возвращается бесконечность.

– prim_mst(n, dist, nodes). Использует алгоритм Прима, чтобы построить минимальное остовное дерево на подмножестве nodes. Возвращает итоговый вес MST или бесконечность, если остов не связен.

4. Приближённые алгоритмы

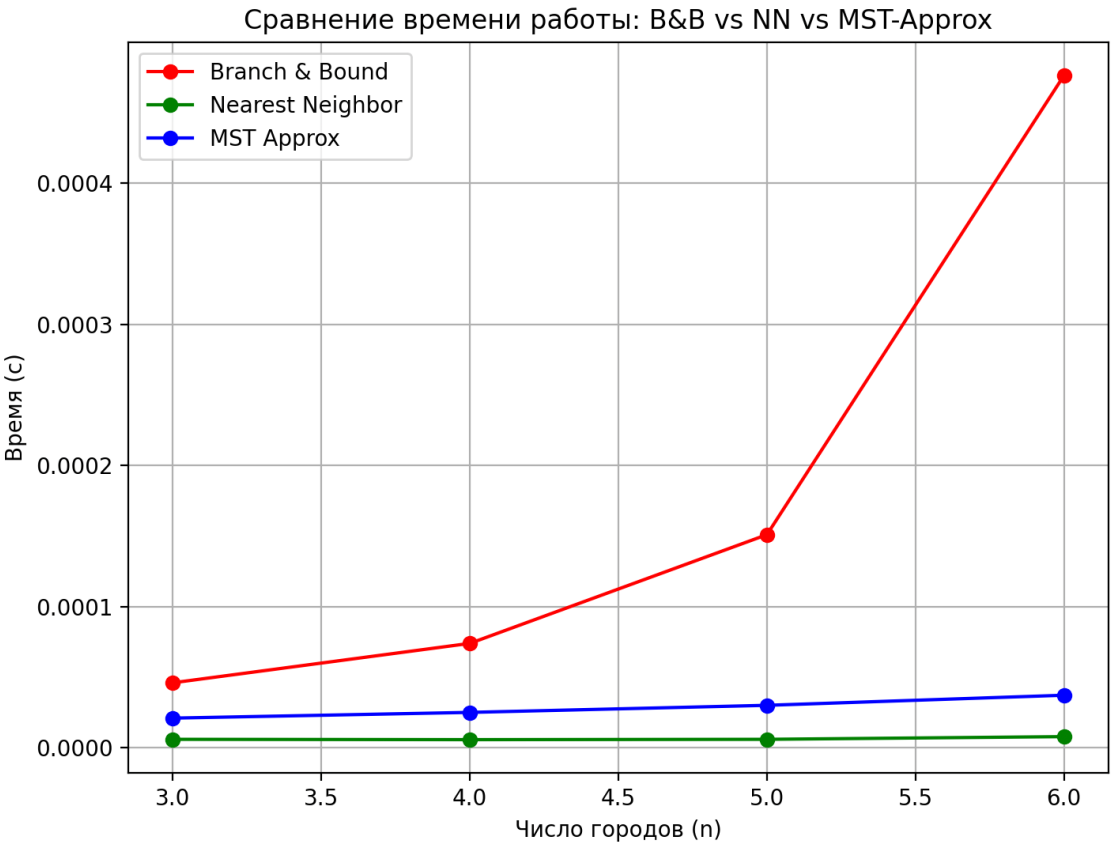
Помимо точного метода ветвей и границ, в коде присутствуют два эвристических решения:

(а) «Ближайший сосед» (Nearest Neighbor). Начинаем с 0, каждый раз выбираем непосещённый город с минимальной дугой от текущего, пока не посетим все, а затем возвращаемся в 0.

(б) «MST + DFS» (AMP). Строится MST по всем вершинам, обходится в глубину. Из полученной последовательности удаляются повторные визиты вершин, а в конце добавляется обратное ребро к 0. Это классический 2-приближённый метод для метрического TSP.

5. Сложность

Формально, ветви и границы для TSP могут рассматривать все перестановки вершин, то есть обладать факториальной оценкой O(n!), ведь в наихудшем случае не сработают эффективные отсечения. Тем не менее, нижние оценки (полусумма двух минимальных рёбер и MST + 2 рёбра) на практике убирают значительную часть ветвей, существенно снижая реальные вычислительные затраты и позволяя решать экземпляры умеренного размера. Что касается памяти, код хранит матрицу dist (n×n), массивы visited, path, min_edges и рекурсивный стек глубиной до n, что даёт $O(n^2)$ пространственной сложности.



Визуализация работы алгоритма

Вывод

Таким образом, в рамках лабораторной работы реализован полноценный метод ветвей и границ с двумя оценками (МВиГ), дающий оптимальное решение, а также два приближённых алгоритма (Nearest Neighbor и MST+DFS), которые работают быстрее, но потенциально могут давать менее качественные решения. Это даёт возможность сравнивать точный и эвристические подходы как по времени, так и по качеству найденного маршрута.