Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

УДК 004

**Отчет по лабораторной работе**

**«Решение задачи коммивояжёра и его визуализация»**

Выполнил:

студент группы РИС-23-1б

Сингур И. С.

Проверил:

Кандидат технических наук

Петренко А. А.

г. Пермь, 2024 г.

# **СОДЕРЖАНИЕ**

[**СОДЕРЖАНИЕ** 2](#_Toc165548351)

[**ВВЕДЕНИЕ** 3](#_Toc165548352)

[**1 АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЁРА** 5](#_Toc165548353)

[**2 РАЗРАБОТКА КОДА НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ С++** 10](#_Toc165548354)

[**2.1 АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ** 10](#_Toc165548355)

[**2.2 НАПИСАНИЕ КОДА ДЛЯ РЕШЕНИЯ ЗАДАЧИ** 11](#_Toc165548357)

[**3. ВИЗУАЛИЗАЦИИ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЁРА** 12](#_Toc165548359)

[**4. ЭКСПЕРИМЕНТАЛЬНАЯ ПРОВЕРКА РАБОТЫ ПРОГРАММЫ** 13](#_Toc165548359)

[**ЗАКЛЮЧЕНИЕ** 14](#_Toc165548360)

[**СПИСОК ИСТОЧНИКОВ** 15](#_Toc165548361)

# **ВВЕДЕНИЕ**

Задача коммивояжёра (Traveling Salesman Problem, TSP) является одной из классических задач комбинаторной оптимизации. В данной задаче необходимо найти кратчайший путь, проходящий через все города из заданного списка ровно один раз и возвращающийся в исходный город. TSP имеет огромное практическое значение в логистике, транспорте, проектировании микросхем, а также в других областях, где требуется оптимизация пути или маршрута.

Решение задачи коммивояжёра также является сложной задачей, требующей применения различных оптимизационных и алгоритмических подходов. Существует множество методов для решения TSP, включая методы полного перебора, приближенные алгоритмы, генетические алгоритмы, муравьиные алгоритмы и другие.

Одним из первых, кто сформулировал ранний вариант этой задачи, был выдающийся ирландский математик, физик и механик XIX в. — Уильям Роуэн Гамильтон. Ученый изучал симметрии икосаэдра (двадцатигранника) и в 1857 году предложил игру «Икосиан» (англ. «Icosian Game»), цель которой заключалась в прохождении всех вершин додекаэдра (двенадцатигранника) ровно по одному разу и только по его ребрам, с последующим возвратом в отправную точку. Иными словами нужно было найти так называемый гамильтонов цикл на графе с 20 узлами, в данном случае являющийся решением задачи и содержащий 20 ребер («двадцать» на древнегреческом «icosa», отсюда и название игры).[1]

Исследование задачи коммивояжёра позволяет не только находить оптимальные или приближенные решения для конкретных ситуаций, но и изучать общие принципы оптимизации пути и маршрутов. Решение TSP также имеет важное значение в разработке алгоритмов для других комбинаторных задач и служит основой для изучения сложных оптимизационных проблем в области искусственного интеллекта и логистики.

Данная лабораторная работа заключается в анализе методов решения задачи коммивояжёра, исследовании их эффективности, изучении возможных оптимизаций, а также она включает в себя поиск наиболее эффективной визуализации решения. Исследование эффективности алгоритмов коммивояжёра требует изучения их скорости работы, точности результата, способности к решению больших задач и т.д.

Таким образом, выполнение данной лабораторной работы способствует развитию не только теоретических знаний, но и практических навыков в области разработки и оптимизации алгоритмов.

Объект исследования: вычислительный процесс нахождения оптимального пути методом ветвей и границ.

Предмет исследования: программа-приложение, определяющая кратчайший путь с помощью метода ветвей и границ для решения задачи коммивояжера.

Цель: реализация программы на языке программирования С++ для решения задачи коммивояжёра, выполнение визуализации решения.

Задачи:

1. Провести анализ алгоритмов решения;
2. Разработать кода на языке программирования С++;
3. Разработать визуализацию решения задачи коммивояжёра.

# **1. АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЁРА**

Перед тем как решать задачу коммивояжёра, мы должны проанализировать все возможные методы решения, выявить их достоинства и недостатки, быстроту, доступность(простоту). Для этой задачи были разработаны следующие алгоритмы:

**1.Полный перебор:**

Полный перебор (или метод «грубой силы») — метод решения задачи путем перебора всех возможных вариантов. Сложность полного перебора зависит от количества всех возможных решений задачи. Если пространство решений очень велико, то полный перебор может не дать результатов в течение нескольких лет или даже столетий.

**Анализ:**

**Cильный стороны:**

1. Гарантированное нахождение оптимального решения: Полный перебор рассматривает все возможные варианты, поэтому гарантированно найдет наилучшее решение.
2. Простота реализации: Алгоритм полного перебора обычно прост в реализации, поскольку он прямолинеен и не требует сложных эвристик.

**Слабые стороны:**

1. Экспоненциальная сложность: С увеличением числа вариантов решения время работы алгоритма возрастает экспоненциально.
2. Неэффективность для больших задач: Для задач с большим пространством решений полный перебор может быть непрактичным из-за его огромных вычислительных требований.

**2.Случайный перебор:**

Обычно выбор решения можно представить последовательностью выборов. Если делать эти выборы с помощью какого-либо случайного механизма, то решение находится очень быстро, так что можно находить решение многократно и запоминать «рекорд», т. е. наилучшее из встретившихся решений. Этот наивный подход существенно улучшается, когда удается учесть в случайном механизме перспективность тех или иных выборов, т. е. комбинировать случайный поиск с эвристическим методом и методом локального поиска. Такие методы применяются, например, при составлении расписаний для Аэрофлота.

**Анализ:**

**Сильные стороны:**

1. Быстрота: Случайный перебор может быстро находить приемлемые решения, особенно если пространство решений велико.
2. Простота: Алгоритм случайного перебора прост в реализации.
3. Возможность улучшения: Случайный перебор можно улучшить, добавив эвристики и методы локального поиска, что делает его более эффективным.

**Слабые стороны:**

1. Не гарантирует оптимальное решение: Случайный перебор не гарантирует нахождения наилучшего решения, он может найти только «хорошее» решение.
2. Зависимость от случайности: Качество решения зависит от случайного выбора, что может привести к непостоянству результатов.
3. Трудность выбора подходящей эвристики: Для эффективного случайного перебора необходимы хорошо подобранные эвристики, которые могут быть сложными в разработке.[2]

3.**Метод ветвей и границ**:

Метод ветвей и границ предложен в 1963 году группой авторов Дж. Литлом, К. Мурти, Д. Суини, К. Кэролом. Широко используемый вариант поиска с возвращением, фактически является лишь специальным частным случаем метода поиска с ограничениями4. Ограничения в данном случае основываются на предположении, что на множестве возможных и частичных решений задана некоторая функция цены и что нужно найти оптимальное решение, т.е. решение с наименьшей ценой. Для применения метода ветвей и границ функция цены должна обладать тем свойством, что цена любого частичного решения не превышает цены любого расширения этого частичного решения (Заметим, что в большинстве случаев функция цены неотрицательна и даже удовлетворяет более сильному требованию).

Столь большой успех применения данного метода объясняется тем, что авторы первыми обратили внимание на широту возможностей метода, отметили важность использования специфики задачи и сами воспользовались спецификой задачи коммивояжера.

В основе метода ветвей и границ лежит идея последовательного разбиения множества допустимых решений на подмножества. На каждом шаге метода элементы разбиения подвергаются проверке для выяснения, содержит данное подмножество оптимальное решение или нет. Проверка осуществляется посредством вычисления оценки снизу для целевой функции на данном подмножестве. Если оценка снизу не меньше рекорда — наилучшего из найденных решений, то подмножество может быть отброшено. Проверяемое подмножество может быть отброшено еще и в том случае, когда в нем удается найти наилучшее решение. Если значение целевой функции на найденном решении меньше рекорда, то происходит смена рекорда. По окончании работы алгоритма рекорд является результатом его работы.

Если удается отбросить все элементы разбиения, то рекорд — оптимальное решение задачи. В противном случае, из неотброшенных подмножеств выбирается наиболее перспективное (например, с наименьшим значением нижней оценки), и оно подвергается разбиению. Новые подмножества вновь подвергаются проверке и т.д.[3]

**Анализ:**

**Сильные стороны:**

1. Гарантированное нахождение оптимального решения (при правильной реализации): Метод гарантирует нахождение оптимального решения, если оно существует, при условии правильного определения нижней оценки.
2. Эффективность: Метод отсекает целые ветви пространства поиска, которые не могут содержать оптимальное решение, что существенно сокращает количество рассматриваемых вариантов.
3. Гибкость: Метод можно адаптировать к различным задачам оптимизации, используя соответствующие функции оценки.

**Слабые стороны:**

1. Сложность реализации: Метод ветвей и границ может быть сложным в реализации, так как требует тщательного выбора функции оценки и стратегии ветвления.
2. Зависимость от функции оценки: Эффективность метода сильно зависит от качества функции оценки. Неточная или слишком слабая оценка может привести к рассмотрению большого количества лишних вариантов.
3. Потенциально большой объем памяти: Метод может требовать значительного объема памяти для хранения информации о ветвях и оценках.

**Вывод:**

Таким образом, для задач с небольшим пространством поиска, где требуется гарантированно найти наилучшее решение, полный перебор может быть вполне приемлемым. Если же пространство поиска обширно и допустимо довольствоваться "хорошим" решением, то случайный перебор, особенно в сочетании с эффективными эвристиками, станет предпочтительным выбором. Для сложных задач оптимизации, требующих оптимального решения и допускающих хорошую функцию оценки, метод ветвей и границ представляет собой наиболее эффективный подход, несмотря на сложность его реализации.

# **2. РАЗРАБОТКА КОДА НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ С++**

# **2.1 АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ**

Для данной лабораторной работы нужен был метод ветвей и границ.

**Алгоритм решения:**

1. Составление матрицы смежности;
2. Нахождение минимума по строкам;
3. Редукция строк;
4. Нахождение минимума по столбцам;
5. Редукция столбцов;
6. Нахождение оценок для нулевых элементов;
7. Редукция матрицы;
8. Выбор: Если мы еще не нашли все отрезки пути, которые позволяют вернуться Коммивояжеру в исходный город, то возвращаемся к шагу Если все отрезки пути найдены или оставшаяся часть очевидна – переходим к заключительному шагу – соединение путей. В реалиях данной задачи необходимо перейти к шагу 2.;
9. Построение маршрута;
10. Вычисление длины пути.

**Вывод:**

Таким образом, чтобы правильно решить задачу коммивояжёра методом ветвей и границ, нужно чётко следовать алгоритму. Написание алгоритма для решения задачи помогает структурировать последующую работу, что облегчает выполнение лабораторной работы.

**2.2 НАПИСАНИЕ КОДА ДЛЯ РЕШЕНИЯ ЗАДАЧИ**

Для того, чтобы написать код следуя выбранному алгоритму, нужно выбрать платформу, на которой будет происходить реализация. Для своей лабораторной работы я буду использовать Visual Studio.

Интегрированная среда разработки Visual Studio является творческой стартовой площадкой, которую можно использовать для редактирования, отладки и сборки кода, а также для публикации приложения. В дополнение к стандартному редактору и отладчику, предоставляемых большинством интегрированных сред разработки, Visual Studio включает компиляторы, средства завершения кода, графические конструкторы и многие другие функции для улучшения процесса разработки программного обеспечения. Visual Studio является самой быстрой интегрированной средой разработки для повышения производительности.[4]

**Вывод:**

Таким образом, выбрав Visual Studio в качестве платформы для своей лабораторной работы, вы получаете доступ к мощному инструменту с широким набором функций, упрощающих процесс разработки программного обеспечения. Visual Studio позволит вам эффективно редактировать, отлаживать и собирать код, реализующий выбранный вами алгоритм оптимизации. Богатый функционал среды, включая компиляторы, средства завершения кода и графические конструкторы, поможет вам сосредоточиться на решении задачи и создании качественного кода.

Код решения задачи коммивояжёра рассмотрен в приложении А. UML таблица рассмотрена в приложении Б.

**3. ВИЗУАЛИЗАЦИЯ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЁРА**

Визуализация играет ключевую роль, преобразуя сложные структуры данных и алгоритмы в интуитивно понятные графические представления. Это позволяет разработчикам анализировать эффективность алгоритмов, выявлять узкие места, оптимизировать производительность и эффективно доносить сложные технические концепции до коллег и пользователей. Для визуализации решения задачи коммивояжёра была использована графическая библиотека OpenGL.

OpenGL является на данный момент одним из самых популярных программных интерфейсов (API) для разработки приложений в области двумерной и трехмерной графики. Стандарт OpenGL был разработан и утвержден в 1992 году ведущими фирмами в области разработки программного обеспечения, а его основой стала библиотека IRIS GL, разработанная Silicon Graphics.

С точки зрения разработчика, OpenGL - это набор команд, которые управляют использованием графической аппаратуры. Если аппаратура состоит только из адресуемого буфера кадра, тогда OpenGL должен быть реализован полностью с использованием ресурсов центрального процессора. Обычно графическая аппаратура предоставляет различные уровни ускорения: от аппаратной реализации вывода линий и полигонов до изощренных графических процессоров с поддержкой различных операций над геометрическими данными.[5]

Визуализация решения задачи коммивояжёра показана в приложении В.

**Вывод:**

В целом, использование OpenGL для визуализации решения задачи коммивояжёра способствует более глубокому пониманию алгоритма, эффективной коммуникации результатов и принятию взвешенных решений.

**4. ЭКСПЕРИМЕНТАЛЬНАЯ ПРОВЕРКА РАБОТЫ ПРОГРАММЫ**

С целью выяснить, как работает программа, какие недостатки есть в её работе, был проведён экспериментальный запуск.

В ходе первого запуска была выявлена одна ошибка при решении задачи коммивояжёра, при построении маршрута путь выходил за границы массива и указывал на пустое отрицательное множество (мусор).

Ошибка была устранена, программа сработала корректно. Ошибка показана в приложении Г.

**Вывод:**

Экспериментальный запуск программы является важным этапом разработки, позволяющим выявить скрытые ошибки и недочеты в работе алгоритма. Несмотря на кажущуюся корректность работы программы, важно проводить тщательное тестирование на различных наборах данных, включая граничные случаи, чтобы гарантировать ее надежность и предотвратить возникновение ошибок в реальных условиях эксплуатации.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения лабораторной работы был разработан программный код, реализующий алгоритм решения задачи коммивояжера. Код успешно прошел тестирование и продемонстрировал корректную работу, находя оптимальный маршрут минимальной длины. Для наглядного представления решения была реализована визуализация с использованием графической библиотеки OpenGL, что позволило детально изучить найденный маршрут, проанализировать его эффективность и выявить возможные оптимизации.

Разработанное программное обеспечение с успехом решает поставленную задачу и может быть использовано для анализа и оптимизации логистических цепочек и транспортных сетей.

**СПИСОК ИСТОЧНИКОВ**

1. <https://galyautdinov.ru/post/zadacha-kommivoyazhera>
2. <https://studfile.net/preview/9535178/page:3/>
3. <https://habr.com/ru/articles/560468/>
4. <https://visualstudio.microsoft.com/ru/#vs-section>
5. <https://www.rsdn.org/article/opengl/ogltutor.xml>

**ПРИЛОЖЕНИЕ А**

**(код для задачи коммивояжёра)**

void answer(int\*\*\* mat, int n, int\*\* help, int\* path)//Эта функция реализует алгоритм решения задачи коммивояжера, используя Венгерский алгоритм.

{

for (int l = 0; l < n; l++)

{

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

{

int min = 1000000;

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

{

if (mat[i][j] && min > \*mat[i][j])

{

min = \*mat[i][j];

}

}

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

{

if (mat[i][j])

{

\*mat[i][j] -= min;

}

}

}

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

{

int min = 1000000;

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

{

if (mat[i][j] && min > \*mat[i][j])

{

min = \*mat[i][j];

}

}

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

{

if (mat[i][j])

{

\*mat[i][j] -= min;

}

}

}

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

{

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

{

help[i][j] = 0;

}

}

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

{

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

{

if (mat[i][j] && !\*mat[i][j])

{

int hmin = 1000000;

int vmin = 1000000;

for (int l = 0; l < n; l++)

{

if (l != i && mat[l][j] && hmin > \*mat[l][j])

{

hmin = \*mat[l][j];

}

}

for (int l = 0; l < n; l++)

{

if (l != j && mat[i][l] && hmin > \*mat[i][l])

{

vmin = \*mat[i][l];

}

}

help[i][j] = hmin + vmin;

}

}

}

int mcost = 0, mi = 0, mj = 0;

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

{

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

{

if (mat[i][j] && mcost < help[i][j])

{

mcost = help[i][j];

mi = i;

mj = j;

}

}

}

path[mi] = mj;

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

{

mat[i][mj] = nullptr;

}

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

{

mat[mi][i] = nullptr;

}

mat[mj][mi] = nullptr;

}

}

void preparation(int\*\*\*& mat, int& n, int\*\*& help, int\*& result)// Эта функция подготавливает данные для алгоритма TSP (коммивояжера)

{

n = amountVerts;// Присваиваем количество вершин из графа

// Выделяем память под вспомогательные матрицы help и result

help = new int\* [n];

result = new int[n];

// Выделяем память под трехмерную матрицу mat, которая будет хранить матрицу смежности графа

mat = new int\*\* [n];

// Инициализируем help

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

{

help[i] = new int[n];

}

// Заполняем mat значениями из матрицы смежности графа

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

{

mat[i] = new int\* [n];

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

{

if (graph.adjMatrix[i][j] == 0)

{

mat[i][j] = nullptr;

continue;

}

mat[i][j] = new int(graph.adjMatrix[i][j]);

}

}

}

void TSP(int\*\*\* mat, int n, int\*\* help, int\* result)// Эта функция является точкой входа для решения задачи коммивояжера (TSP).

{

preparation(mat, n, help, result);

int s = 0;

answer(mat, n, help, result);

cout << endl << "Отрезки путей: ";

for (int i = 0, j = 0; i < n; i++)

{

j = result[i];

cout << i + 1 << " -> " << j + 1 << '\t';

s += graph.adjMatrix[i][j];

}

cout << endl;

cout << endl << "Кратчайший путь: ";

int tmp = 0;

for (int l = 0; l < n;)

{

for (int i = 0, j = 0; i < n; i++)

{

if (tmp == 0 || i + 1 == tmp)

{

if (tmp == 0)

{

cout << i + 1;

}

j = result[i];

tmp = j + 1;

if (tmp > 0)

{

cout << " -> " << tmp;

}

l++;

}

}

}

cout << endl << "Минимальное расстояние: " << s;

cout << endl;

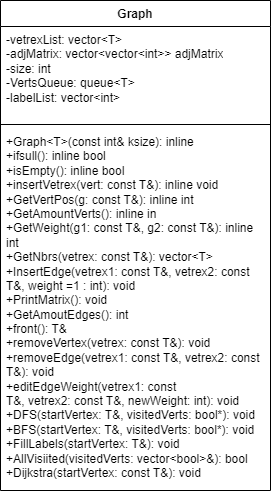
}

Полный код с визуализацией:

<https://drive.google.com/file/d/1Ku683_veCMl_iR0E-xUJXAW-HZ1owJxe/view?usp=sharing>

**ПРИЛОЖЕНИЕ Б**

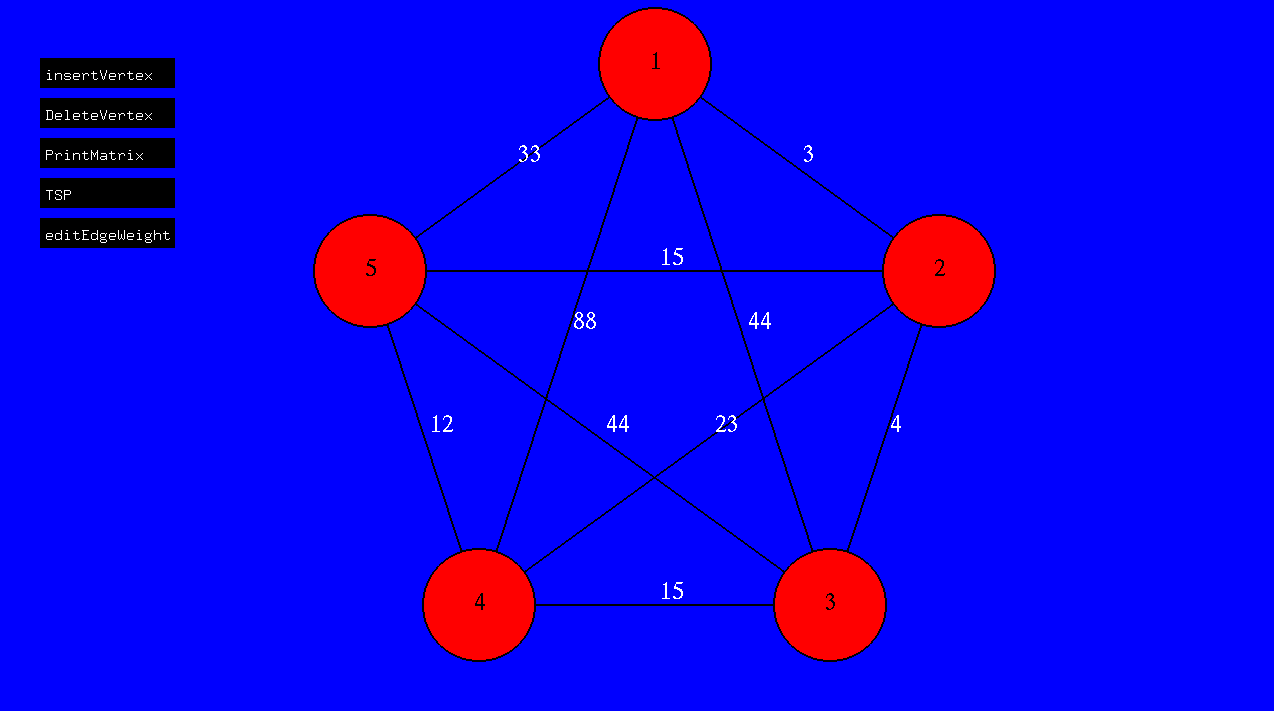
**(UML)**



**ПРИЛОЖЕНИЕ В**

**(Визуализация)**





**ПРИЛОЖЕНИЕ Г**

**(Ошибка)**

