Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет
по лабораторной работе № 6

Дисциплина: Анализ алгоритмов

Тема: Муравьиный алгоритм

Студент: Платонова Ольга

Группа: ИУ7-55Б

Преподаватели: Волкова Л. Л.
Строганов Ю. В.

Москва, 2020 г.

Оглавление

Введение

В данной лабораторной работе требуется изучить и реализовать алгоритмы решения задачи коммивояжера: муравьиный алгоритм и алгоритм полного перебора. Также необходимо выполнить тестирование и анализ алгоритмов.

1. Аналитическая часть

В данном разделе будут рассмотрены понятия, связанные с задачей коммивояжера, а также сформулированы основные принципы алгоритмов, ее реализующих.

1.1. Задача коммивояжера

Одна из самых известных и важных задач транспортной логистики – задача коммивояжера или «задача о странствующем торговце». Также встречается название «задача китайского почтальона».

Суть задачи сводится к поиску оптимального (кратчайшего, быстрейшего или самого дешевого) пути, проходящего через промежуточные пункты по одному разу и возвращающегося в исходную точку. К примеру, нахождение наиболее выгодного маршрута, позволяющего коммивояжеру посетить со своим товаром определенные города по одному разу и вернуться обратно. Мерой выгодности маршрута может быть минимальное время поездки, минимальные расходы на дорогу или минимальная длина пути. В наше время, когда стоимость доставки часто бывает сопоставима со стоимостью самого товара, а скорость доставки — один из главных приоритетов, задача нахождения оптимального маршрута приобретает огромное значение.

Для решения задачи коммивояжера ее надо представить как математическую модель. При этом исходные условия можно записать в формате матрицы — таблицы, где строкам соответствуют города отправления, столбцам — города прибытия, а в ячейках указываются расстояния (время, стоимость) между ними; или в виде графа — схемы, состоящей из вершин (точек, кружков), которые символизируют города, и соединяющих их ребер (линий), длина которых соответствует расстоянию между городами. [1]

1.2. Полный перебор

Общепризнанно, что задача коммивояжёра в общем случае гарантированно решается оптимально только полным перебором всех вариантов. Однако с ростом количества узлов время, затрачиваемое на полный перебор, растёт экспоненциально. Таким образом, на машине с четырёхядерным процессором 2,67 ГГц 10 узлов обсчитывается в среднем за 5 миллисекунд, 20 узлов – за 15 минут, а на расчёт оптимального пути для 60 узлов уйдёт более 6 триллионов

лет… В алгоритмической реализации полного перебора возможно применение двух условий, сокращающих время перебора и отсекающих заведомо неоптимальные пути, а именно:

1. При построении очередного варианта пути подсчитывать его длину и, если эта длина превышает уже найденный локальный минимум длины – пропускать этот вариант и все те, которые он порождает.

2. Если следующее выбранное ребро пересекает одно из ранее построенных рёбер – пропустить этот вариант и все те, которые он порождает. [2]

1.3. Муравьиный алгоритм

Муравьиный алгоритм — эвристический метод, основанный на моделировании поведения муравьев, ищущих пути от своей колонии к источникам пищи. Первую версию такого алгоритма предложил доктор наук Марко Дориго в 1992 году. Этот метод позволяет относительно быстро найти хорошее, но не обязательно оптимальное решение.

В основе алгоритма лежит поведение муравьиной колонии — маркировка более удачных путей большим количеством феромона. Работа начинается с размещения муравьёв в вершинах графа (городах), затем начинается движение муравьёв — направление определяется вероятностным методом, на основании формулы вида:

$$P_i = \frac{l_i^q * f_i^p}{\sum_{k=0}^{N} l_k^q * f_k^p}, \quad (1)$$

где P_i — вероятность перехода по пути i,

l_i — величина, обратная весу (длине) i-го перехода,

f_i — количество феромона на i-м переходе,

q — величина, определяющая «жадность» алгоритма,

p — величина, определяющая «стадность» алгоритма.

В реальном мире муравьи ходят в случайном порядке и по нахождении продовольствия возвращаются в свою колонию, прокладывая феромонами тропы. Если другие муравьи находят такие тропы, они, вероятнее всего, пойдут по ним. Вместо того, чтобы отслеживать

цепочку, они укрепляют её при возвращении, если в конечном итоге находят источник питания. Таким образом, когда один муравей находит путь от колонии до источника пищи, другие муравьи, скорее всего пойдут по этому пути, и положительные отзывы в конечном итоге приводят всех муравьёв к одному, кратчайшему, пути.

1.4. Вывод

В данном разделе были рассмотрены понятия, связанные с задачей коммивояжера, а также сформулированы основные принципы алгоритмов, ее реализующих.

Задача коммивояжера является, возможно, наиболее исследуемой задачей комбинаторной оптимизации. Неудивительно, что именно она была выбрана первой для решения с использованием подхода, заимствующего механизмы поведения муравьиной колонии. Позже этот подход нашел применение в решении многих других комбинаторных проблем, в числе которых задачи о назначениях, задача раскраски графа, задачи маршрутизации, задачи из областей data mining и распознавания образов и другие.

Эффективность муравьиных алгоритмов сравнима с эффективностью общих метаэвристических методов, а в ряде случаев – и с проблемно-ориентированными методами. Наилучшие результаты муравьиные алгоритмы показывают для задач с большими размерностями областей поиска. Муравьиные алгоритмы хорошо подходят для применения вместе с процедурами локального поиска, позволяя быстро находить начальные точки для них.

2. Конструкторская часть

В данном разделе будет рассмотрены схемы алгоритмов решения задачи коммивояжера.

2.1. Разработка алгоритмов

На рисунке 2.1 изображена схема *алгоритма полного перебора с рекурсией*.

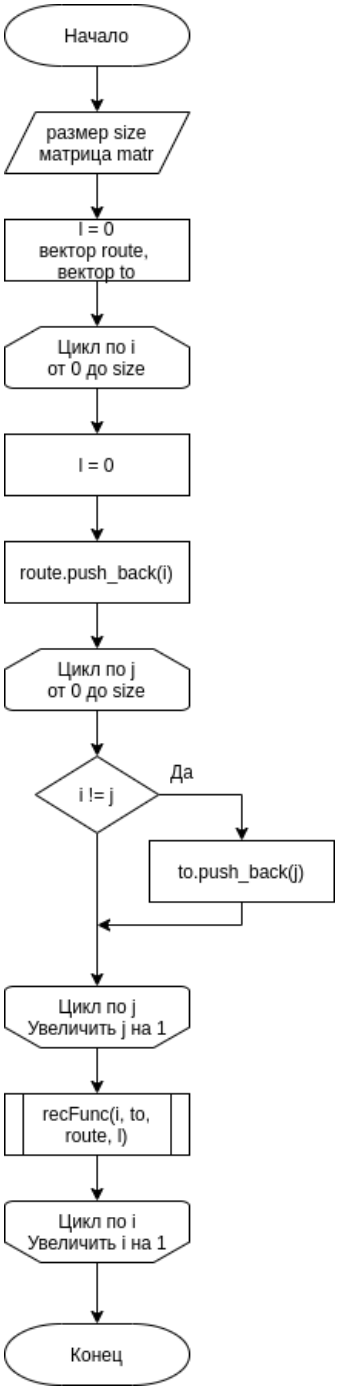

Рисунок 2.1. Схема алгоритма полного перебора.

На рисунке 2.2 изображена схема муравьиного алгоритма.

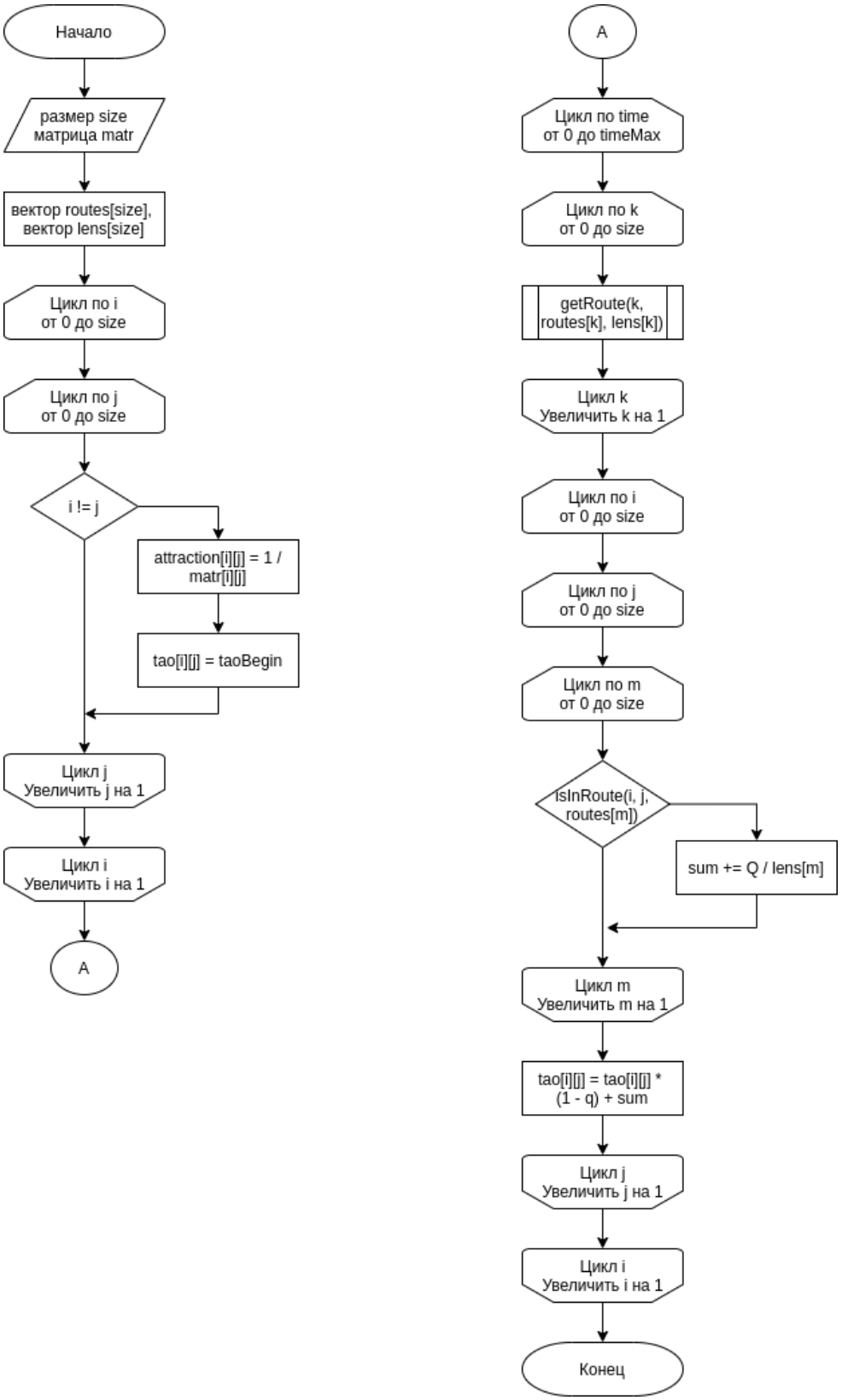

Рисунок 2.2. Схема муравьиного алгоритма.

2.2. Вывод

В данном разделе были рассмотрены схемы алгоритмов решения задачи коммивояжера: полного перебора и муравьиного алгоритма.

3. Технологическая часть

В данном разделе будет рассмотрен язык программирования, среда разработки, требуемые инструменты для реализации. Также будет представлена реализация алгоритма.

3.1. Требования к программному обеспечению

I. Программа должна предусматривать чтение данных из файла.
II. В результате обработки полученные данные должны быть записаны в файл.
III. Должны быть произведены замеры времени для каждой из реализаций.

3.2. Средства реализации

В данной работе используется язык программирования C++ из-за опыта написания на нем. Среда разработки – Qt.

Для замеров процессорного времени использовалась функция clock(). [3]

3.3. Реализация алгоритмов

Реализация алгоритма полного перебора с рекурсией представлена в листинге 3.1.

Листинг 3.1. Алгоритм полного перебора.

```
#include "BruteForce.h"

void bruteForce(size_t size, vector<vector<size_t>> matr)
{
    L_MIN = MAX;
    ROUTE_MIN.clear();

    size_t l = 0;
    vector<size_t> route;
    ROUTE_MIN.resize(size + 1);
    vector<size_t> to(size - 1);

    for (size_t i = 0; i < size; i++) {
        route.clear();
        to.clear();
        l = 0;

        route.push_back(i);
        for (size_t j = 0; j < size; j++) {
            if (j != i) {
                to.push_back(j);
            }
```

```
            }
            recFunc(i, to, matr, route, l);
        }

        cout << endl << "ROUTE: ";
            outputVector(ROUTE_MIN);
        cout << "LENGHT: " << L_MIN << endl << endl;
}

void recFunc(size_t from, vector<size_t> to, vector<vector<size_t>> matr,
             vector<size_t> route, size_t l)
{

    size_t n = to.size();
    size_t last;
    if (n == 1) {
        last = to[0];

        route.push_back(last);
        l += matr[from][last];
        l += matr[last][route[0]];
        route.push_back(route[0]);

        if (l < L_MIN) {
            L_MIN = l;
            ROUTE_MIN = route;
        }
        return;
    }

    vector<size_t> cur_to(n - 1);
    vector<size_t> cur_route(n);
    size_t cur_l;
    for (size_t i = 0; i < n; i++) {
        last = to[i];
        cur_to = cutVector(to, last);
        cur_route = route;
        cur_route.push_back(last);
        cur_l = l + matr[route[route.size() - 1]][last];
        recFunc(last, cur_to, matr, cur_route, cur_l);
    }

}
```

Реализация муравьиного алгоритма представлена в листинге 3.2.

Листинг 3.2. Муравьиный алгоритм.

```
#include "AntAlgorithm.h"

void ant(size_t size, vector<vector<size_t>> matr, size_t alpha, size_t beta,
double q,
         size_t time_max, ofstream& file)
{
    vector<vector<size_t>> routes(size);
    vector<size_t> lens(size);

    vector<vector<double>> attraction(size);
    vector<vector<double>> tao(size);

    for (size_t i = 0; i < size; i++) {
        attraction[i].resize(size);
        tao[i].resize(size);
        lens[i] = 0;
        all[i] = i;

        for (size_t j = 0; j < size; j++) {
            if (i != j) {
                attraction[i][j] = 1.0 / matr[i][j];
                tao[i][j] = tao_start;
            }
        }
    }

    for (size_t time = 0; time < time_max; time++) {
        for (size_t k = 0; k < size; k++) {
            getRoute(all, k, routes[k], lens[k], matr, tao, attraction, alpha,
beta);

            if (lens[k] < L_MIN) {
                L_MIN = lens[k];
                ROUTE_MIN = routes[k];
            }
        }

        for (size_t i = 0; i < size; i++) {
            for (size_t j = 0; j < size; j++) {
                double sum = 0;

                for (size_t m = 0; m < size; m++) {
                    if (isInRoute(i, j, routes[m])) {
                        sum += Q / lens[m];
                    }
                }

                tao[i][j] = tao[i][j] * (1 - q) + sum;
                if (tao[i][j] < tao_min) {
                    tao[i][j] = tao_min;
                }
            }
        }
    }

    outputResAnt(ROUTE_MIN);
```

```
}

void getRoute(vector<size_t> all, size_t start,
              vector<size_t> &route, size_t &len,
              vector<vector<size_t>>    matr,    vector<vector<double>>    tao,
vector<vector<double>> attraction,
              size_t alpha, size_t beta)
{
    route.resize(0);
    route.push_back(start);

    vector<size_t> to = cutVector(all, start);
    size_t n_1 = tao.size() - 2;
    size_t from;

    double coin = 0, sum = 0;
    bool flag = true;

    for (size_t i = 0; i < n_1; i++) {
        sum = 0;
        flag = true;
        from = route[i];

        vector<double> p = vecProbability(from, to, tao, attraction, alpha, beta);
        coin = double(rand() % 10000) / 10000;

        for (size_t j = 0; j < p.size() && flag; j++) {
            sum += p[j];
            if (coin < sum) {
                route.push_back(to[j]);
                len += matr[from][to[j]];
                to = cutVector(to, to[j]);
                flag = false;
            }
        }
    }

    len += matr[route[route.size() - 1]][to[0]];
    route.push_back(to[0]);

    len += matr[route[route.size() - 1]][route[0]];
    route.push_back(route[0]);
}

bool isInRoute(size_t a, size_t b, vector<size_t> route)
{
    bool res = false;
    size_t m = route.size() - 1;

    for (size_t i = 0; i < m && !res; i++) {
        if (a == route[i] && b == route[i + 1]) {
            res = true;
        }
    }
    return res;
}

vector<double> vecProbability(size_t from, vector<size_t> to,
                              vector<vector<double>> tao, vector<vector<double>>
attraction,
                              size_t alpha, size_t beta)
```

```
{
    double znam = 0, chisl = 0;

    size_t n = to.size();
    vector<double> result(n);

    for (size_t i = 0; i < n; i++) {
        znam += pow(tao[from][to[i]], alpha) * pow(attraction[from][to[i]],
beta);
    }

    for (size_t j = 0; j < n; j++) {
        chisl = pow(tao[from][to[j]], alpha) * pow(attraction[from][to[j]],
beta);
        result[j] = chisl / znam;
    }

    return result;
}
```

3.4. Описание тестирования

Для проверки корректности программы необходимо предусмотреть различные наборы переменного числа вершин, включая простейшие наборы.

3.5. Вывод

В данном разделе были рассмотрены инструменты, необходимые для реализации алгоритмов решения задачи коммивояжера, а также были представлены непосредственно реализации.

4. Экспериментальная часть

В данном разделе будут рассмотрены примеры работы программы, произведено тестирование и контроль выполнения, выполнены замеры времени, а также выполнен анализ полученных данных.

4.1. Технические характеристики устройства.

Операционная система – LinuxMint 64-bit;

Память – 8 ГБ;

Процессор – Intel™ Core™ i3-7100U CPU @ 2.40 ГГц;

Логических процессов – 4.

4.2. Сравнительный анализ времени работы алгоритмов

Были проведены 2 серии экспериментов с замером времени на графах разных размеров. В первой серии увеличивался размер графа, во второй были произведены замеры на графе малых размеров. Результаты экспериментов представлены на рисунках 4.1 – 4.2.

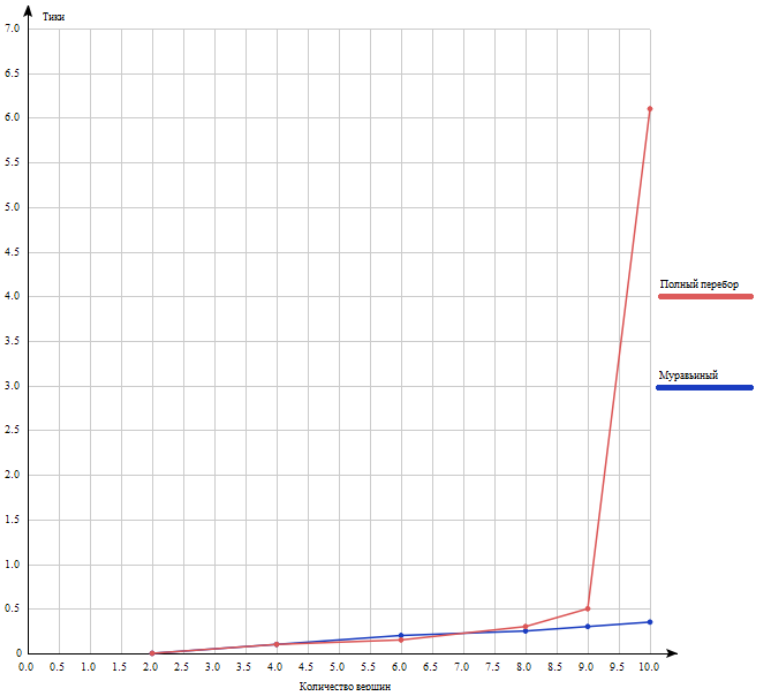

Рисунок 4.1. График зависимости времени работы алгоритмов от размера графа.

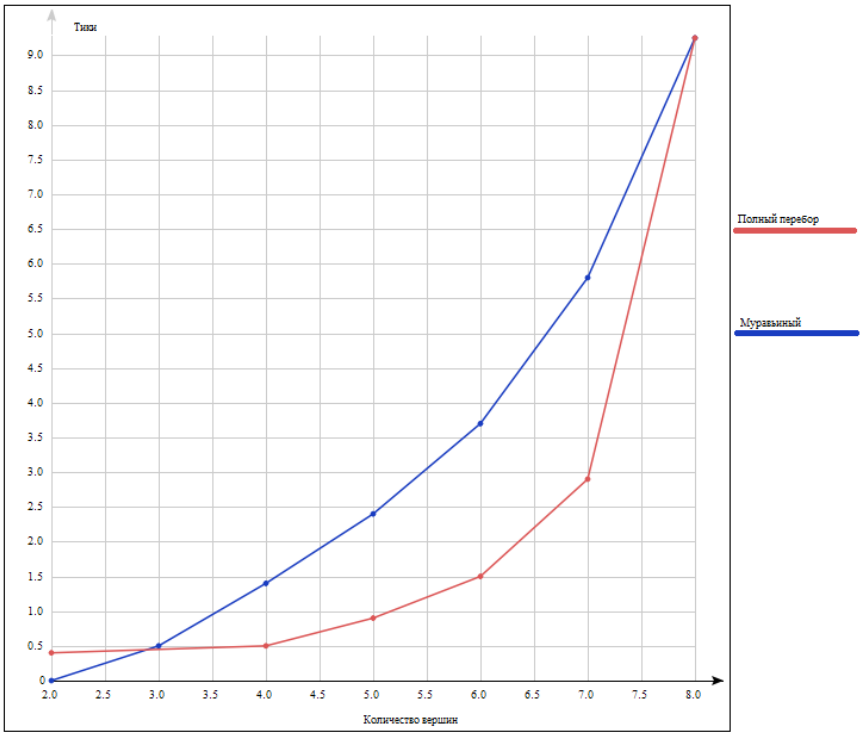

Рисунок 4.2. График зависимости времени работы алгоритмов от размера графа. Малые размеры графа.

Следует отметить, что на графах больших размеров (более 9) муравьиный алгоритм превосходит алгоритм полного перебора по времени. Однако на малых размерах (до 8 вершин) алгоритм полного перебора работает быстрее.

4.3. Вывод

В данном разделе были рассмотрены примеры работы программы, произведено тестирование и контроль выполнения, выполнены замеры времени, а также выполнен анализ полученных данных.

Заключение

Цель работы достигнута, все поставленные задачи выполнены: были разработаны и реализованы алгоритмы решения задачи коммивояжера: полный перебор и муравьиный, а также проведено тестирование и анализ их работы.

В результате анализа было установлено, что на графах малых размеров преимущество по времени работы имеет алгоритм полного перебора. Максимальный выигрыш составляет 2.5 раза на графах размером 6 вершин. Что касается графов больших размеров, то муравьиный алгоритм имеет выигрыш по времени, который достигает 30 раз на графах размером 10 вершин.

Список литературы

1. Галяутдинов Р.Р. Задача коммивояжера - метод ветвей и границ. URL: http://galyautdinov.ru/post/zadacha-kommivoyazhera (дата обращения: 12.12.2020).
2. Левитин А. В. Глава 3. Метод грубой силы: Задача коммивояжёра // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 159—160. — 576 с. — ISBN 978-5-8459-0987-9
3. Техническая документация.-URL: https://docs.microsoft.com/ru-ru/cpp/c-runtime-library/reference/clock?view=vs-2019 (дата обращения: 20.09.2020). Текст: электронный.