**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»

ДИСЦИПЛИНА «Анализ алгоритмов»

# Лабораторная работа № 6

**Тема** «Муравьиный алгоритм и метод полного перебора для решения задачи коммивояжера»

**Студент** Чалый А.А.

**Группа** ИУ7 – 52Б

**Оценка (баллы)** _____

**Преподаватели** Волкова Л.Л.
Строганов Ю.В.

Москва.
2020 г.

**Оглавление**

## Введение

Задача коммивояжера — одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанный города хотя бы по одному разу с последующим возвратом в исходный город. [1]

Задача коммивояжера относится к числу транс вычислительных: уже при относительно небольшом числе городов (66 и более) она не может быть решена методом перебора вариантов никаким теоретически мыслимыми компьютерами за время, меньшее нескольких миллиардов лет.

Муравьиный алгоритм — один из эффективных полиномиальных алгоритмов для нахождения приближенных решений задачи коммивояжера, а также решения аналогичных задач поиска маршрутов на графах. Суть подхода заключается в анализе и использовании модели поведения муравьев, ищущих пути от колонии к источнику питания, и представляет собой метаэвристическую оптимизацию. [2]

Целью данной работы является изучение муравьиного алгоритма на материале решения задачи коммивояжера.

Задачи работы:

1) Описать методы решения.
2) Описать реализацию, реализованный метод.
3) Выбрать класс данных и составить набор данных.
4) Провести параметризацию метода на основании муравьиного алгоритма для выбранного класса данных.
5) Провести сравнительный анализ двух методов.
6) Дать рекомендации о применимости метода решения задачи коммивояжера на основе муравьиного алгоритма.

# 1. Аналитическая часть

В данном разделе содержится описание задачи коммивояжера и методы ее решения.

## 1.1 Описание задачи

В общем случае задача коммивояжера может быть сформулирована следующим образом: найти самый выгодный (самый короткий, самый дешевый, и т.д.) маршрут, начинающийся в исходном городе и проходящий ровно один раз через каждый из указанных городов, с последующим возвратом в исходный город.

Проблему коммивояжера можно представить в виде модели на графе, то есть, используя вершины и ребра между ними. Таким образом, M вершин графа соответствуют M городам, а ребра (*i, j*) между вершинами *i* и *j* — пути сообщения между этими городами. Каждому ребру (*i, j*) можно сопоставить критерий выгодности маршрута $c_{ij} >= 0$, который можно понимать как, например, расстояние между городами, время или стоимость поездки. В целях упрощения задачи и гарантии существования маршрута обычно считается, что модельный граф задачи является полностью связным, то есть, что между произвольной парой вершин существует ребро.

Гамильтоновым циклом называется маршрут, включающий ровно по одному разу каждую вершину графа. Таким образом, решение задачи коммивояжера — это нахождение гамильтонова цикла минимального веса в полном взвешенном графе.

## 1.2 Метод полного перебора

Пусть дано M —число городов, D — матрица смежности, каждый элемент которой — вес пути из одного города в другой. Существует метод грубой силы решения поставленной задачи, а именно полный перебор всех возможных гамильтоновых циклов в заданном графе с нахождением минимального по весу. Этот метод гарантированно даст идеальное решение

(глобальный минимум по весу). Однако стоит учитывать, что сложность такого алгоритма составляет M! и время выполнения программы, реализующий такой подход, будет расти экспоненциально в зависимости от размеров входной матрицы.

### 1.3 Муравьиный алгоритм

На практике чаще всего необходимо получить решение как можно быстрее, при этом требуемое решение не обязательно должно быть наилучшим, был разработан ряд методов, называемых эвристическими, которые решают поставленную задачу за гораздо меньшее время, чем метод полного перебора. В основе таких методов лежат принципы из окружающего мира, которые в дальнейшем могут быть формализованы.

Одним из таких методов является муравьиный метод. Он применим к решению задачи коммивояжера и основан на идее муравейника.

Введем математическую модель. У муравья есть 3 чувства:

- зрение (муравей может оценить длину ребра);
- обоняние (муравей может унюхать феромон — вещество, выделяемое муравьем, для коммуникации с другими муравьями);
- память (муравей запоминает свой маршрут).

Благодаря введению обоняния между муравьями возможен непрямой обмен информацией.

Введем вероятность $P_{k,\,ij}(t)$ выбора следующего города j на маршруте муравьем k, который в текущий момент времени t находится в городе i.

$$p_{i,j} = \frac{(r_{i,j}^{a})(n_{i,j}^{b})}{\sum (r_{i,j}^{a})(n_{i,j}^{b})}, \qquad (1)$$

где

$r_{i,j}$ — феромон на ребре ij;

$n_{i,j}$ — привлекательность города j;

a — параметр влияния длины пути;

b — параметр влияния феромона.

Очевидно, что при b = 0 алгоритм превращается в классический жадный алгоритм, а при a = 0 он быстро сойдется к некоторому субоптимальному решению. Выбор правильного соотношения параметров является предметом исследований, и в общем случае производится на основании опыта.

После того, как муравей успешно проходит маршрут, он оставляет на всех пройденных ребрах феромон, обратно пропорциональный длине пройденного пути:

$$\Delta r_{k,ij} = \frac{Q}{L_k}, \qquad (2)$$

где

$Q$ — количество феромона, переносимого муравьем;

$L_k$ — стоимость k—го пути муравья (обычно длина).

После окончания условного дня наступает условная ночь, в течение которой феромон испаряется с ребер коэффициентом p. Количество феромона на следующий день вычисляется по следующей формуле:

$$r_{ij}\,(t + 1) = (1 - p)r_{i,j}(t) + \Delta r_{i,j}(t), \qquad (3)$$

где

$p_{i,j}$ — доля феромона, который испарится;

$r_{i,j}(\mathrm{t})$ — количество феромона на дуге ij;

$\Delta r_{i,j}(\mathrm{t})$ — количество отложенного феромона.

Таким образом, псевдокод муравьиного алгоритма можно представить так:

1) Ввод матрицы расстояний D, количества городов M;
2) Инициализация параметров алгоритма — a, b, Q, tmax. p;
3) Инициализация ребер —присвоение “привлекательности” $n_{i,j}$ и начальной концентрации феромона $r_{start}$;
4) Размещение муравьев в случайно выбранные города без совпадений;
5) Инициализация начального кратчайшего маршрута $L_p$ = null и определение длины кратчайшего маршрута $L_{min}$ = inf;
6) Цикл по времени жизни колонии t = 1, tmax;
   a) Цикл по всем муравьям k = 1, M;
      i) Построить маршрут $T_k(t)$ по правилу 1.1 и рассчитать длину получившегося маршрута $L_k(t)$;
      ii) Обновить феромон на маршруте по правилу 1.2;
      iii) Если $L_k(t) < L_{min}$, то $L_{min} = L_k(t)$ и $L_p = T_k(t)$;
   b) Конец цикла по муравьям;
   c) Цикл по всем ребрам графа;
      i) Обновить следы феромона на ребре по правилу 1.3;
   d) Конец цикла по ребрам;
7) Конец цикла по времени;

8) Вывести кратчайший маршрут $L_p$ и его длину $L_{min}$.

**Вывод**

Таким образом, существуют две группы методов для решения задачи коммивояжера — точные и эвристические. К точным относится метод полного перебора, к эвристическим — муравьиный метод. Применение муравьиного алгоритма обосновано в тех случаях, когда необходимо быстро найти решение или когда для решения задачи достаточно получения первого приближения. В случае необходимости максимально точного рещения используется алгоритм полного перебора.

## 2. Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов решения задачи коммивояжера.

### 2.1 Схемы алгоритмов

На рис. 1 приведена схема основной части алгоритма полного перебора.

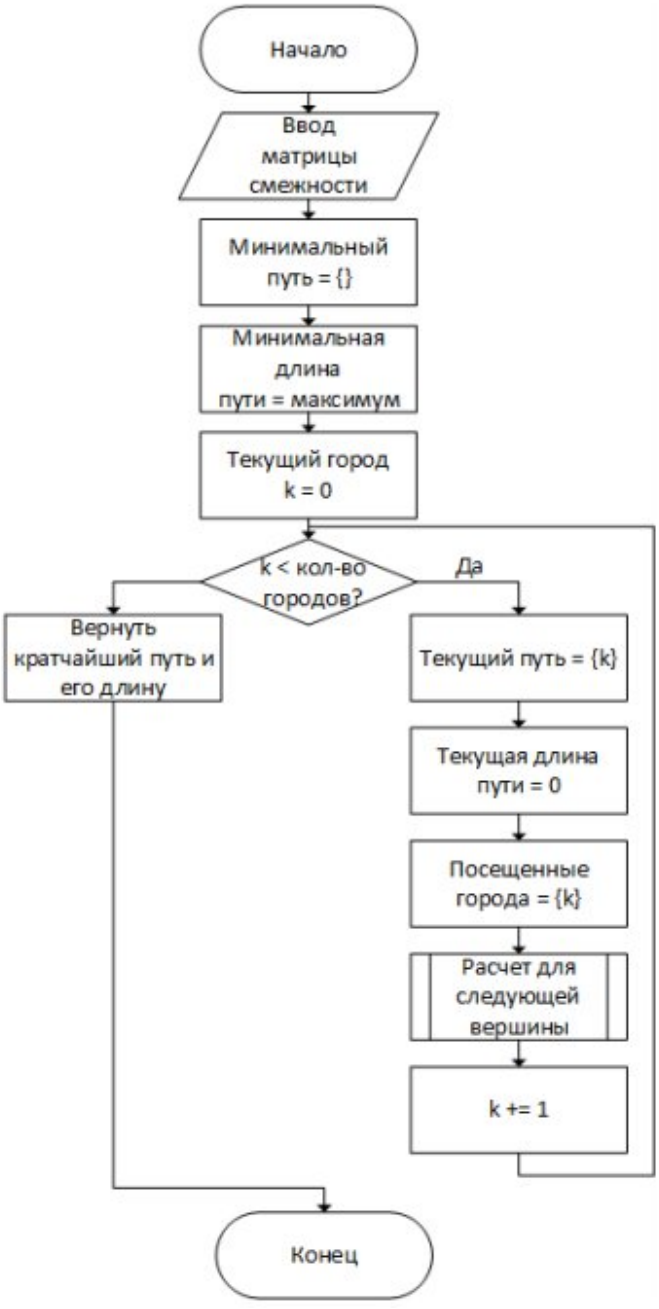


Рисунок 1. Схема основной части алгоритма полного перебора

На рис. 2 приведена схема рекурсивной части алгоритма полного перебора.

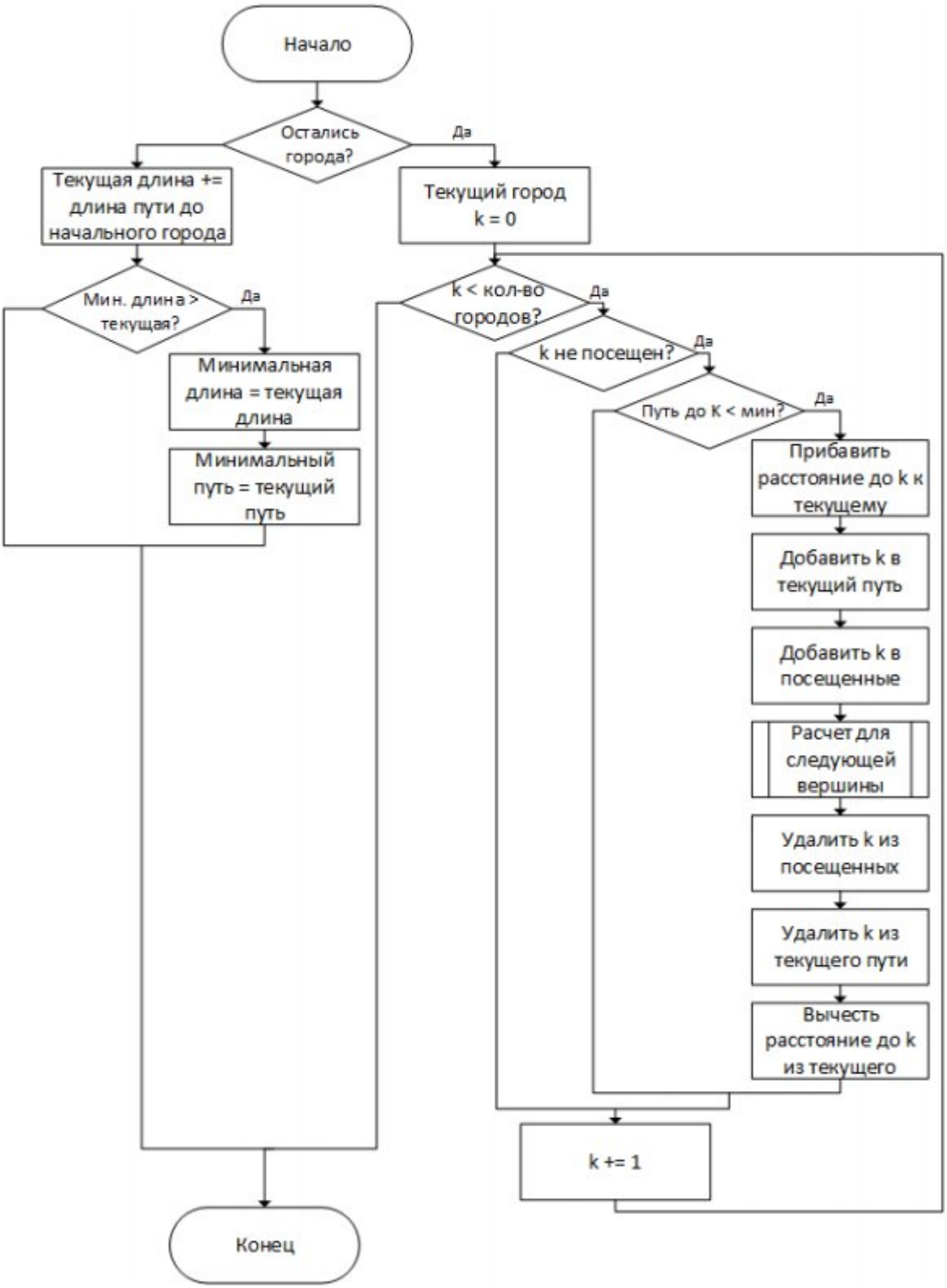


Рисунок 2. Схема рекурсивной части алгоритма полного перебора

На рис. 3 и 4 приведена схема муравьиного алгоритма.

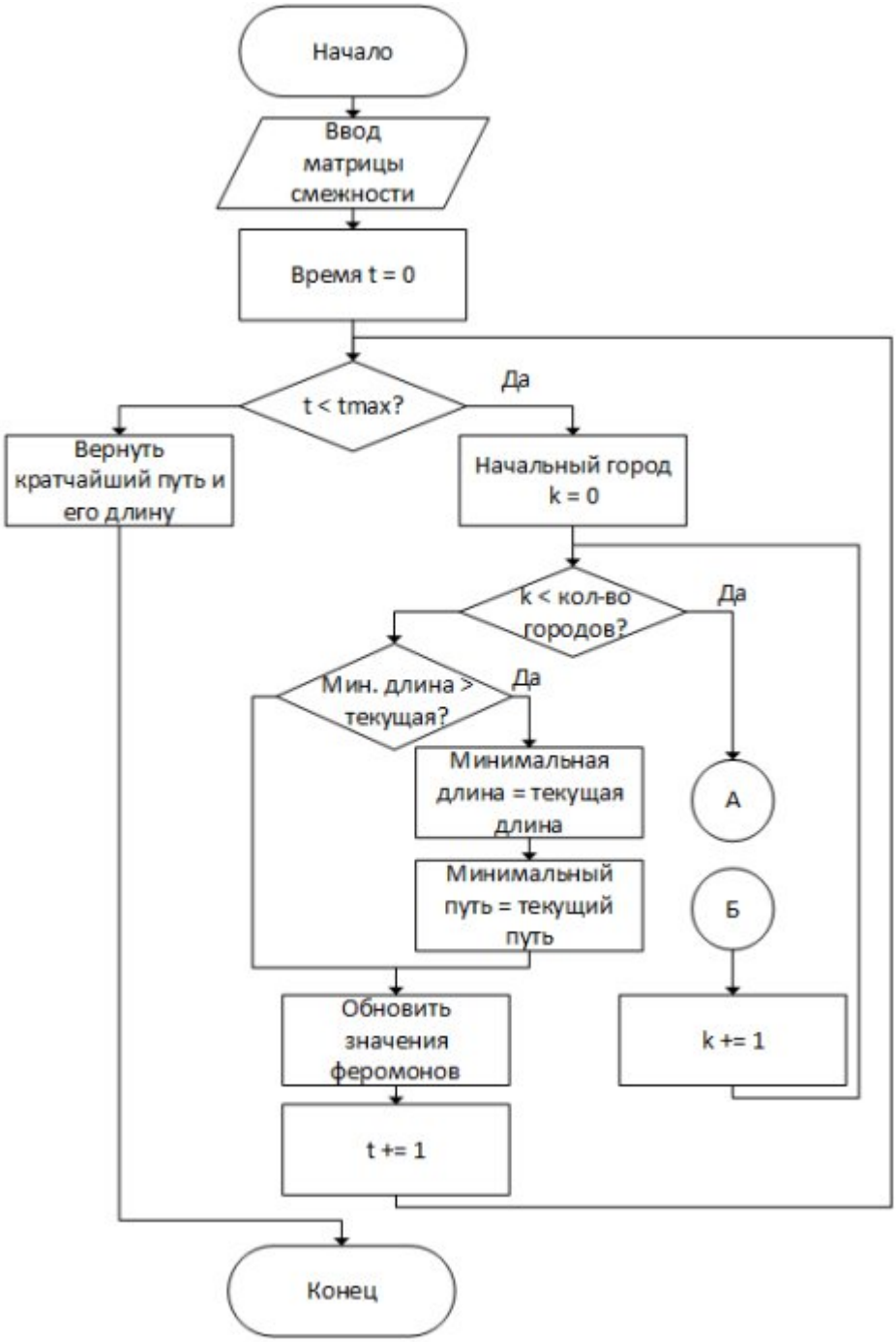


Рисунок 3. Схема муравьиного алгоритма ч. 1

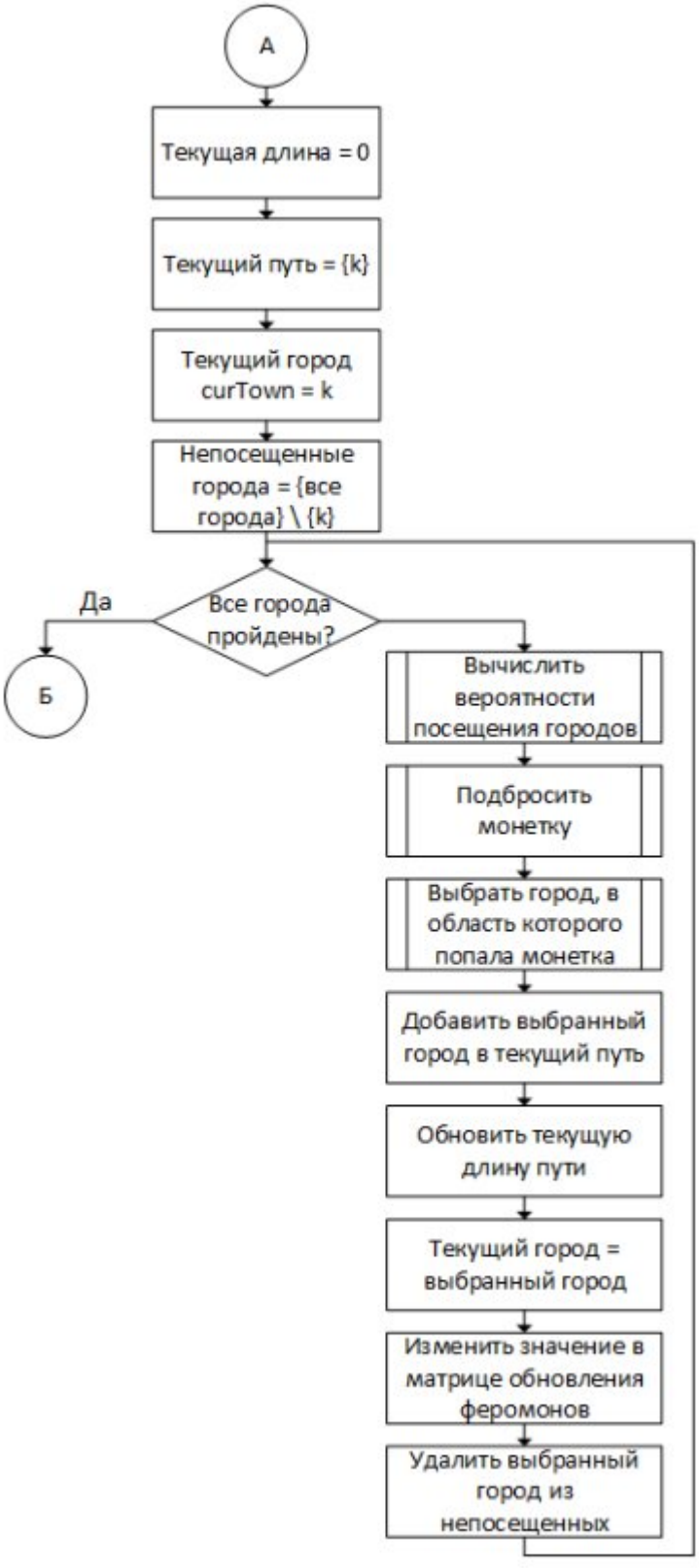


Рисунок 4. Схема муравьиного алгоритма ч. 2

**Вывод**

Как видно из схем алгоритмов, кол-во блоков операций в алгоритме полного перебора меньше, чем в муравьином алгоритме. Это объясняется

более сложной моделью данных в случае муравьиного метода. При этом стоит отметить, что в стандартном алгоритме полного перебора внесено одно изменение, а именно проверка текущей длины пути: если она уже больше длины минимального на текущий момент маршрута, то дальше маршрут по этому пути не строится. Такая модификация позволяет увеличить быстродействие программы.

## 3.Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации и представлен листинг кода.

### 3.1 Требования к программному обеспечению

Входные данные: Матрица смежности графа.

Выходные данные: Последовательность целых чисел — минимальный по стоимости маршрут и действительное число — суммарная стоимость этого маршрута..

Функциональная схема решения задачи коммивояжера в нотации IDEF0 представлена на рис. 5.

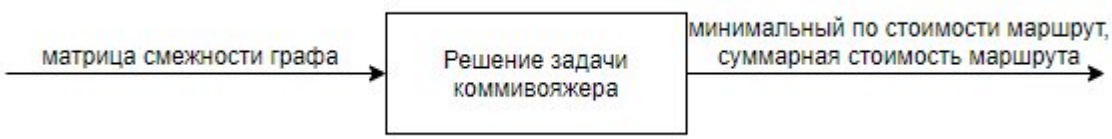


Рисунок 5. Функциональная схема решения задачи коммивояжера

### 3.2 Средства реализации

В качестве языка программирования был выбран C++ т.к. данный язык часто использовался мною. Данный язык позволяет писать высокоуровневый абстрактный код, который при этом работает со скоростью близкой к машинному коду.

Среда разработки – QtCreator, которая предоставляет умную проверку кода, быстрое выявление ошибок и оперативное исправление, вкупе с автоматическим рефакторингом кода и богатыми возможностями в навигации.

Время работы было замерено с помощью функции steady_clock() из библиотеки chrono . [3]

Для тестирования использовался компьютер на базе процессора Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz, 3401 МГц, ядер: 4, логических процессоров: 4.

### 3.3 Листинг кода

В данном пункте представлен листинг кода функций. В листинге 1 и 2 представлена реализация алгоритма полного перебора, состоящая из основной функции BruteForce и рекурсивной Hamilton. В листинге 3 представлена реализация муравьиного алгоритма.

Листинг 1. Реализация основной части алгоритма полного перебора

```
Path_info bruteForce(const Matrix<double> &distances)
{
    int size = distances.size();
    vector<bool> visited(size, false);
    vector<int> curr_path;
    Path_info Respath;
    Respath.length = MAX_NUM;
    double curLen = 0;
    for (int i = 0; i < size; i++)
    {
        visited[i] = true;
        curr_path.clear();
        curr_path.push_back(i);
        curLen = 0;
        Hamilton(distances, Respath, curr_path,
visited, curLen);
    }
    return Respath;
}
```

Листинг 2. Реализация рекурсивной части алгоритма полного перебора

```
void Hamilton(const Matrix<double> &distances,
Path_info &Respath, vector<int> &curr_path,
vector<bool> &visited, double &curLen)
```

```
{
    if (curr_path.size() == distances.size())
    {
        double tmp =
distances[curr_path.back()][curr_path[0]];
        if (curLen + tmp < Respath.length)
        {
            Respath.path = curr_path;
            Respath.length = curLen + tmp;
        }
        return;
    }
    for (int i = 0; i < (int) distances.size(); i++)
    {
        if (!visited[i])
        {
            double tmp =
distances[curr_path.back()][i];
            if (curLen + tmp > Respath.length)
                continue;
            curLen += tmp;
            curr_path.push_back(i);
            visited[i] = true;
            Hamilton(distances, Respath, curr_path,
visited, curLen);
            visited[i] = false;
            curr_path.pop_back();
            curLen -= tmp;
        }
    }
}
```

Листинг 3. Реализация муравьиного алгоритма.

```
Path_info ant(const Matrix<double> &distances, const
int &tMax, const double &alpha, const double &p)
{
    const size_t size = distances.size();

    const double q = distances.average() * size;
    const double betta = 1 - alpha;
```

```
    Matrix <double> attractions(distances);
    attractions.inverse();

    Matrix<double> teta(size, START_TETA);

    Matrix<double> deltaTeta(size);

    Path_info minRoute;
    minRoute.length = -1;
    std::vector<double> probabilities(size, 0.0);

    for (int t = 0; t < tMax; t++)
    {

        deltaTeta.zero();

        for (int k = 0; k < (int) size; k++)
        {
            std::vector<int> curPath = {k};
            double curLength = 0;
            int curTown = k;

            std::vector<int> unvisited =
getUnvisited(curPath, size);
            while (curPath.size() != size)
            {
                fill(probabilities.begin(),
probabilities.end(), 0.0);

                for (const auto &town : unvisited)
                {
                    int index = getIndex(unvisited,
town);
                    if
(fabs(distances[curTown][town]) > EPS)
                    {
                        double sum = 0;
                        for (auto n : unvisited)
```

```
                                sum +=
pow(teta[curTown][n], alpha) *
pow(attractions[curTown][n], betta);
                            probabilities[index] =
pow(teta[curTown][town], alpha) *
pow(attractions[curTown][town], betta) / sum;
                        }
                        else
                            probabilities[index] = 0;
                    }

                    double coin = tossCoin();
                    size_t townIndex = 0;
                    double curProbability = 0;
                    for (size_t j = 0; j < size; j++)
                    {
                        curProbability += probabilities[j];
                        if (coin < curProbability)
                        {
                            townIndex = j;
                            break;
                        }
                    }
                    int chosenTown = unvisited[townIndex];
                    curPath.push_back(chosenTown);
                    curLength +=
distances[curTown][chosenTown];
                    deltaTeta[curTown][chosenTown] += q /
curLength;
                    curTown = chosenTown;
                    unvisited.erase(unvisited.begin() +
townIndex);
                }

                curLength +=
distances[curPath[curPath.size() - 1]][curPath[0]];
                if (minRoute.length < -EPS || (curLength <
minRoute.length))
                {
                    minRoute.length = curLength;
```

```
                    minRoute.path = curPath;
                }
            }

            teta *= (1.0 - p);
            teta += deltaTeta;
            teta.replaceZero(MIN_TETA);
        }
        return minRoute;
    }
```

**Вывод**

В данном разделе была представлена реализация муравьиного алгоритма и алгоритма полного перебора, решающие задачу коммивояжера, для дальнейшего сравнительного анализа точности и скорости вычислений.

# 4. Экспериментальная часть

В данном разделе приведены примеры работы программы, проведена параметризация метода решения задачи коммивояжера на основе муравьиного алгоритма, а также проведен сравнительный анализ двух алгоритмов.

## 4.1 Примеры работы

На рисунках 6 - 7 приведены примеры работы программы. В консоль выводится входная матрица смежности, длина кратчайшего маршрута и сам маршрут для двух алгоритмов.

```
   | 0  1  2
-------------
0  | 0  3  7
1  | 3  0  3
2  | 7  3  0

Bruteforce algorythm result:
 D = 13
 path: 0 1 2
Ant algorythm result:
 D = 13
 path: 0 2 1
```

Рисунок 6. Пример работы при матрице 3X3

```
   | 0  1  2  3  4  5  6  7  8  9  10
-----------------------------------
0  | 0  3  4  2  6  6  7  1  1  3  9
1  | 3  0  3  3  2  1  5  6  7  8  1
2  | 4  3  0  4  5  4  3  2  9  2  3
3  | 2  3  4  0  9  2  3  7  8  8  6
4  | 6  2  5  9  0  6  7  2  2  3  7
5  | 6  1  4  2  6  0  6  3  10 9  8
6  | 7  5  3  3  7  6  0  6  3  2  7
7  | 1  6  2  7  2  3  6  0  4  1  3
8  | 1  7  9  8  2  10 3  4  0  9  6
9  | 3  8  2  8  3  9  2  1  9  0  2
10 | 9  1  3  6  7  8  7  3  6  2  0

Bruteforce algorythm result:
 D = 20
 path: 0 3 5 1 10 2 6 9 7 4 8
Ant algorythm result:
 D = 21
 path: 9 7 0 8 4 1 5 3 6 2 10
```

Рисунок 7. Пример работы при матрице 11X11

Как видно из приведенных примеров, с увеличением размерности матрицы, муравьиный алгоритм начинает показывать меньшую точность, в то время как полный перебор находит наилучший вариант в любом случае.

### 4.2 Параметризация метода

Для проведения экспериментов была использована матрица смежности 10X10, изображенная на рисунке 8.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | 4 | 2 | 6 | 6 | 7 | 1 | 1 | 3 | 9 |
| 1 | 3 | 0 | 3 | 3 | 2 | 1 | 5 | 6 | 7 | 8 | 1 |
| 2 | 4 | 3 | 0 | 4 | 5 | 4 | 3 | 2 | 9 | 2 | 3 |
| 3 | 2 | 3 | 4 | 0 | 9 | 2 | 3 | 7 | 8 | 8 | 6 |
| 4 | 6 | 2 | 5 | 9 | 0 | 6 | 7 | 2 | 2 | 3 | 7 |
| 5 | 6 | 1 | 4 | 2 | 6 | 0 | 6 | 3 | 10 | 9 | 8 |
| 6 | 7 | 5 | 3 | 3 | 7 | 6 | 0 | 6 | 3 | 2 | 7 |
| 7 | 1 | 6 | 2 | 7 | 2 | 3 | 6 | 0 | 4 | 1 | 3 |
| 8 | 1 | 7 | 9 | 8 | 2 | 10 | 3 | 4 | 0 | 9 | 6 |
| 9 | 3 | 8 | 2 | 8 | 3 | 9 | 2 | 1 | 9 | 0 | 2 |
| 10 | 9 | 1 | 3 | 6 | 7 | 8 | 7 | 3 | 6 | 2 | 0 |

Рисунок 8. Матрица для параметризации метода

В каждом эксперименте фиксировались значения a, b, p и tmax. В течение экспериментов значения a, b, p менялись от 0 до 1 с шагом 0.25, tmax от 10 до 310 с шагом 50. Количество повторов каждого эксперимента равнялось 100, результатом проведения эксперимента считалась усредненная разница между длиной маршрута, рассчитанного алгоритмом полного перебора и муравьиным алгоритмом с текущими параметрами.

На рисунке 9 представлены 10 лучших результатов по наименьшему отклонению от минимального расстояния.

```
0.75,0.75,310,0
0.5,0.5,310,0.01
0.5,0.75,210,0.01
0.75,0.5,310,0.01
0.75,0.75,260,0.01
0.25,0.5,260,0.02
0.5,0.5,210,0.02
0.25,0.75,160,0.03
0.5,0.5,260,0.03
0.25,0.75,110,0.04
```

Рисунок 9. Лучшие результаты

Как видно из представленный таблицы, наилучшим значением настроенного параметра a является значение, равное 0.75, а коэффициента испарения p — 0.75. При tmax = 310 достигается наименьшее различие с минимальным путем.

### 4.3 Сравнительный анализ на материале экспериментальных данных

В рамках данной части были проведен эксперимент, описанный ниже.

Сравнение времени работы алгоритма муравья и полного перебора. Создаются матрицы размерностью от 5 до 13 с шагом 2 заполненные случайными числами. Эксперимент по измерению одной размерности ставился 100 раз.

На рис. 10 представлено время работы алгоритмов при разных размерностях.

```
Matrix size : 5
BruteForce time: 0.997000
Ant algorythm time: 1861.539000
Matrix size : 7
BruteForce time: 28.447000
Ant algorythm time: 6224.429000
Matrix size : 9
BruteForce time: 408.242000
Ant algorythm time: 15918.417000
Matrix size : 11
BruteForce time: 8664.892000
Ant algorythm time: 34588.216000
Matrix size : 13
BruteForce time: 148652.657000
Ant algorythm time: 71268.524000
```

Рисунок 10. Время работы алгоритмов при разных размерностях

Как можно наблюдать на рис. 10 время работы алгоритма полного перебора растет экспоненциально в зависимости от размера матрицы смежности, в отличие от муравьиного алгоритма, время которого изменяется практически линейно.

**Вывод**

В результате проведенных экспериментов были выявлены оптимальные параметры для муравьиного алгоритма: a = 0.75, p = 0.75, tmax = 310. Однако стоит учитывать, что чем больше значение tmax, тем больше вероятность того, что будет найден идеальный маршрут, но при этом будет возрастать время выполнения программы. Также был проведен сравнительный анализ муравьиного алгоритма и алгоритма полного перебора. Алгоритм полного перебора рационально использовать для матриц небольшого размера и в случае, если необходимо получить наименьшее расстояние. В остальных случаях муравьиный алгоритм является более эффективным по времени.

**Заключение**

В ходе работы были изучены и реализованы такие алгоритмы для решения задачи коммивояжера как муравьиный и полного перебора. Был проведен их сравнительный анализ, в ходе которого были получены зависимости времени выполнения алгоритмов от размеров входной матрицы расстояний, кроме того, была проведена параметризация муравьиного метода на фиксированном наборе данных. В результате экспериментов было получено, что муравьиный алгоритм работает быстрее алгоритма полного перебора и линейно зависит от размеров входной матрицы, также были выявлены оптимальные параметры для муравьиного алгоритма, обеспечивающие минимальное отклонение длины вычисленного маршрута от длины минимального. Все поставленные задачи были выполнены. Целью лабораторной работы являлось решение задачи коммивояжера, что также было достигнуто.

## Список использованных источников


1. Мудров В.И. Задача о коммивояжере. — М.: Наука, 1969. — 62 с.
2. Дориго М. Муравьиная система. — М.: Наука, 1996. — 29 с.
3. Официальный сайт Microsoft, документация [электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/cpp/standard-library/chrono?view=vs-2017, свободный (Дата обращения: 10.11.20)