**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 по курсу "Анализ алгоритмов"

**Тема** Муравьиный алгоритм

**Студент** Пересторонин П.Г.

**Группа** ИУ7-53Б

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

Москва — 2020 г.

# Оглавление

**Введение** **2**

**1 Аналитическая часть** **3**

1.1 Задача коммивояжера . . . . . 3

1.2 Алгоритм полного перебора для решения задачи коммивояжера . . . . . 3

1.3 Муравьиный алгоритм для решения задачи коммивояжера . 4

**2 Конструкторская часть** **6**

2.1 Схема алгоритма полного перебора . . . . . 6

**3 Технологическая часть** **12**

3.1 Требования к ПО . . . . . 12

3.2 Средства реализации . . . . . 12

3.3 Листинги кода . . . . . 12

3.4 Тестирование функций. . . . . 18

**4 Исследовательская часть** **20**

4.1 Технические характеристики . . . . . 20

4.2 Пример работы программы . . . . . 20

4.3 Исследование скорости работы алгоритмов. . . . . 22

4.4 Постановка эксперимента . . . . . 23

4.4.1 Код тестирования . . . . . 23

4.4.2 Класс данных 1 . . . . . 25

4.4.3 Класс данных 2 . . . . . 27

**Заключение** **32**

**Литература** **33**

# Введение

Цель работы - изучение муравьиного алгоритма и его применение для решения $NP$-трудных задач эвристическими методами на примере задачи коммивояжёра.

В рамках данной лабораторной работы требуется решить следующие задачи:

- изучить и реализовать алгоритм полного перебора для решения задачи коммивояжера;
- изучить и реализовать муравьиный алгоритм для решения задачи коммивояжера;
- провести параметризацию муравьиного алгоритма на двух классах данных;
- провести сравнительный анализ скорости работы реализованных алгоритмов;
- подготовить отчёт по проведенной работе.

# 1 Аналитическая часть

В данном разделе описаны задача коммивояжёра, а также алгоритм полного перебора и муравьиный алгоритм для решения данной задачи.

## 1.1 Задача коммивояжера

Коммивояжёр (фр. *commis voyageur*) — бродячий торговец. Задача коммивояжёра — важная задача транспортной логистики, отрасли, занимающейся планированием транспортных перевозок [1]. В описываемой задаче рассматривается несколько городов и матрица попарных расстояний между ними. Требуется найти такой порядок посещения городов, чтобы суммарное пройденное расстояние было минимальным, каждый город посещался ровно один раз и коммивояжер вернулся в тот город, с которого начал свой маршрут. Другими словами, во взвешенном полном графе требуется найти гамильтонов цикл минимального веса.

## 1.2 Алгоритм полного перебора для решения задачи коммивояжера

Алгоритм полного перебора для решения задачи коммивояжера предполагает рассмотрение всех возможных путей в графе и выбор наименьшего из них.

Такой подход гарантирует точное решение задачи, однако, так как задача относится к числу $NP$-сложных [2], то уже при небольшом числе городов решение за приемлемое время невозможно.

## 1.3 Муравьиный алгоритм для решения задачи коммивояжера

Муравьиные алгоритмы представляют собой новый перспективный метод решения задач оптимизации, в основе которого лежит моделирование поведения колонии муравьев [3]. Колония представляет собой систему с очень простыми правилами автономного поведения особей.

Каждый муравей определяет для себя маршрут, который необходимо пройти на основе феромона, который он ощущает, во время прохождения, каждый муравей оставляет феромон на своем пути, чтобы остальные муравьи могли по нему ориентироваться. В результате при прохождении каждым муравьем различного маршрута наибольшее число феромона остается на оптимальном пути.

Самоорганизация колонии является результатом взаимодействия следующих компонентов:

- случайность — муравьи имеют случайную природу движения;
- многократность — колония допускает число муравьев, достигающее от нескольких десятков до миллионов особей;
- положительная обратная связь — во время движения муравей откладывает феромон, позволяющий другим особям определить для себя оптимальный маршрут;
- отрицательная обратная связь — по истечении определенного времени феромон испаряется;
- целевая функция.

Пусть муравей обладает следующими характеристиками:

- память — запоминает маршрут, который прошел;
- зрение — определяет длину ребра;
- обоняние — чувствует феромон.

Введем целевую функцию $\eta_{ij} = 1/D_{ij}$, где $D_{ij}$ — расстояние из пункта $i$ до пункта $j$.

Посчитаем вероятности перехода в заданную точку по формуле (1.1):

$$P_{k,ij} = \begin{cases} \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{q \in J_{i,k}}^{m} \tau_{iq}^{\alpha}\eta_{iq}^{\beta}}, \text{вершина не была посещена ранее муравьем k,} \\ 0, \text{иначе} \end{cases} \tag{1.1}$$

где $\alpha, \beta$ – настраиваемые параметры, $J_{i,k}$ - список городов, которые надо посетить $k$-ому муравью, находящемуся в $i$-ом городе, $\tau$ - концентрация феромона, а при $\alpha = 0$ алгоритм вырождается в жадный.

Когда все муравьи завершили движение происходит обновление феромона по формуле (1.2):

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \Delta\tau_{ij}, \Delta\tau_{ij} = \sum_{k=1}^{N} \tau_{ij}^{k} \tag{1.2}$$

где

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k, \text{ребро посещено k-ым муравьем,} \\ 0, \text{иначе} \end{cases} \tag{1.3}$$

$L_k$ — длина пути k-ого муравья, $Q$ — некоторая константа порядка длины путей, $N$ — количество муравьев.

# Вывод

Были рассмотрены задача Коммивояжера, муравьиный алгоритм и алгоритм полного перебора для решения данной задачи.

# 2 Конструкторская часть

В данном разделе представлены схемы муравьиного алгоритма и алгоритма полного перебора для решения задачи коммивояжера.

## 2.1 Схема алгоритма полного перебора

На рисунке 2.1 представлена схема алгоритма полного перебора решения задачи Коммивояжера. На рисунке 2.2 представлена схема нахождения всех перестановок. На рисунках 2.3 и 2.4 представлена схема реализации муравьиного алгоритма для решения задачи коммивояжера.

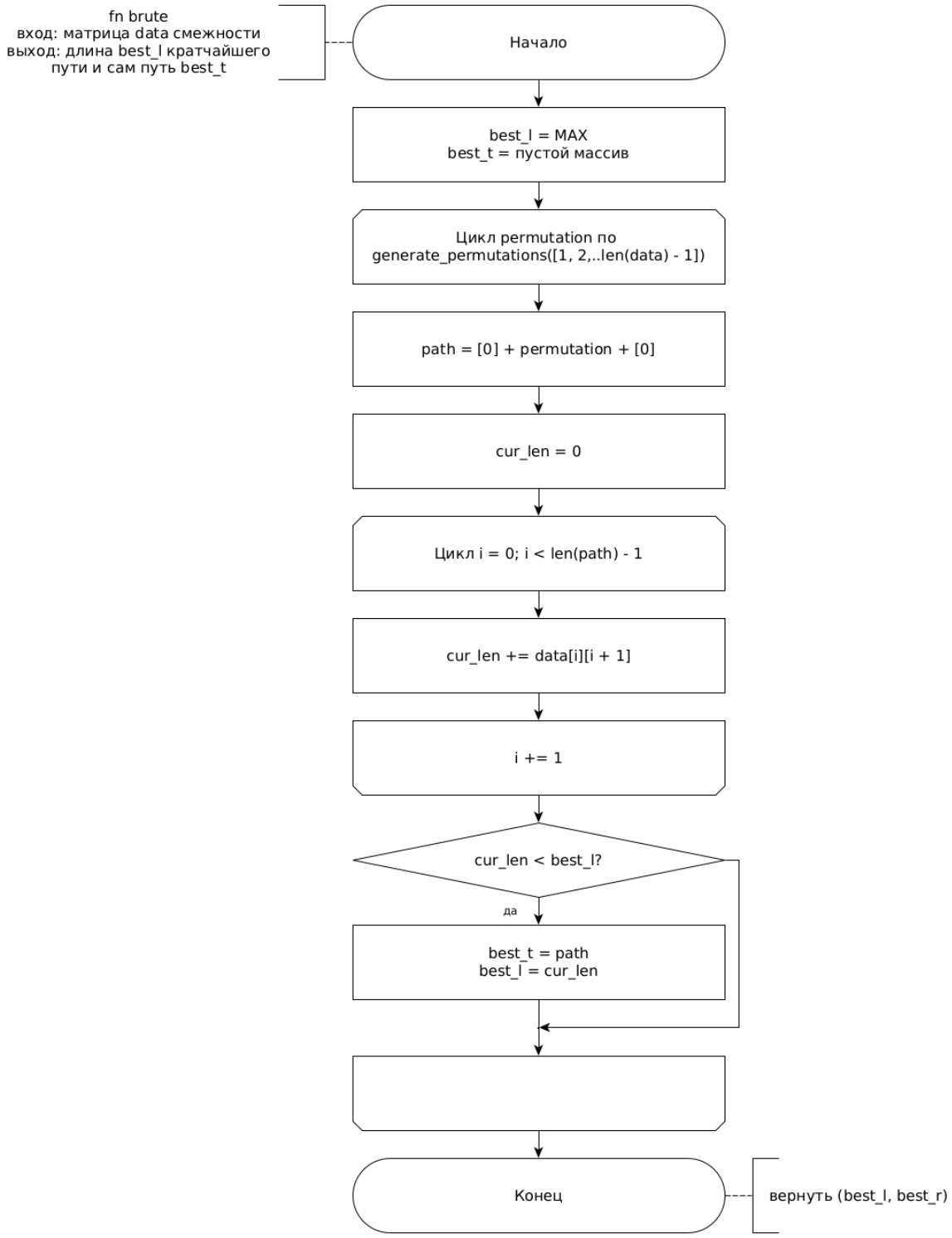


Рис. 2.1: Схема алгоритма полного перебора.

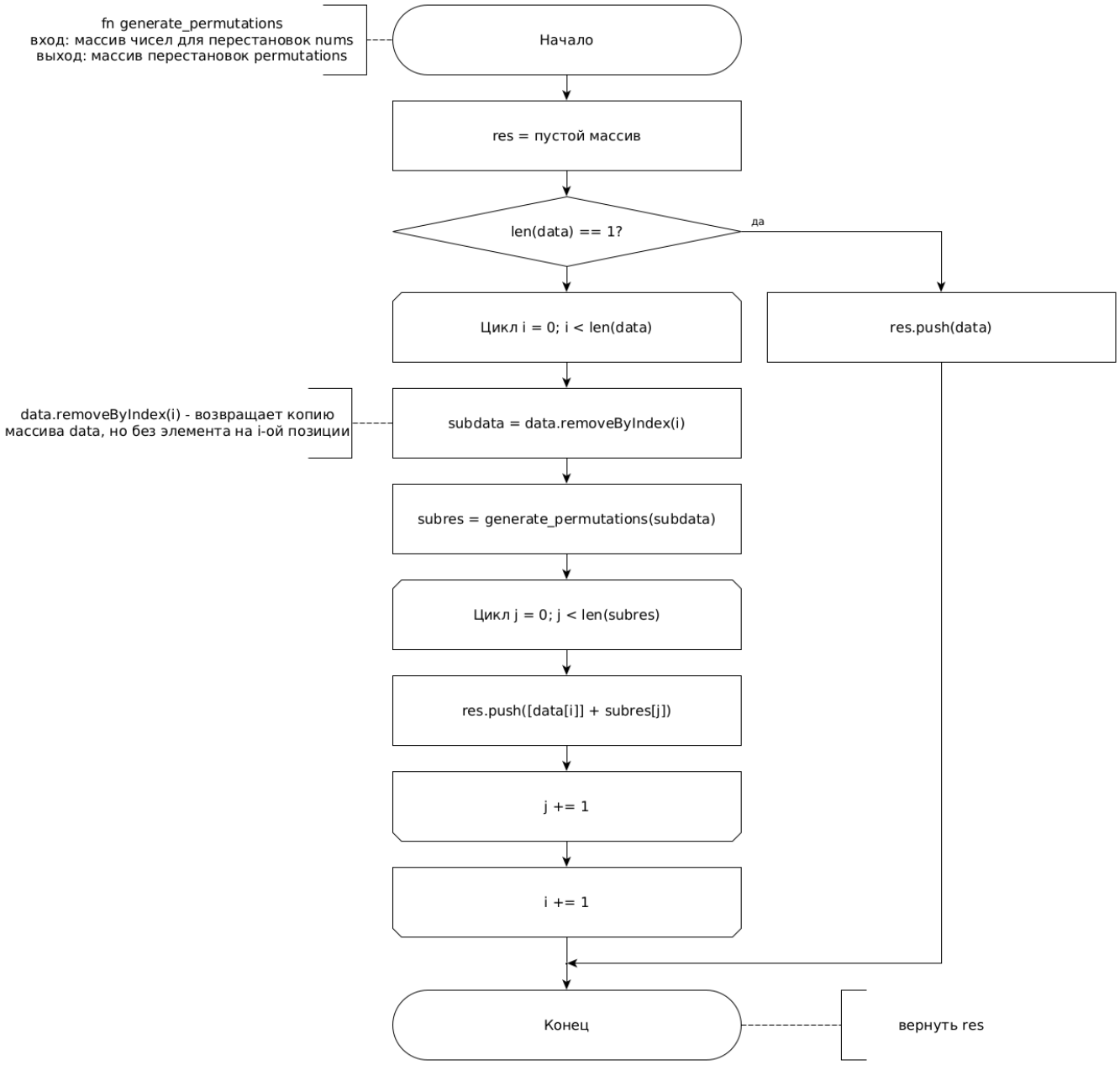


Рис. 2.2: Схема алгоритма нахождения всех перестановок в графе.

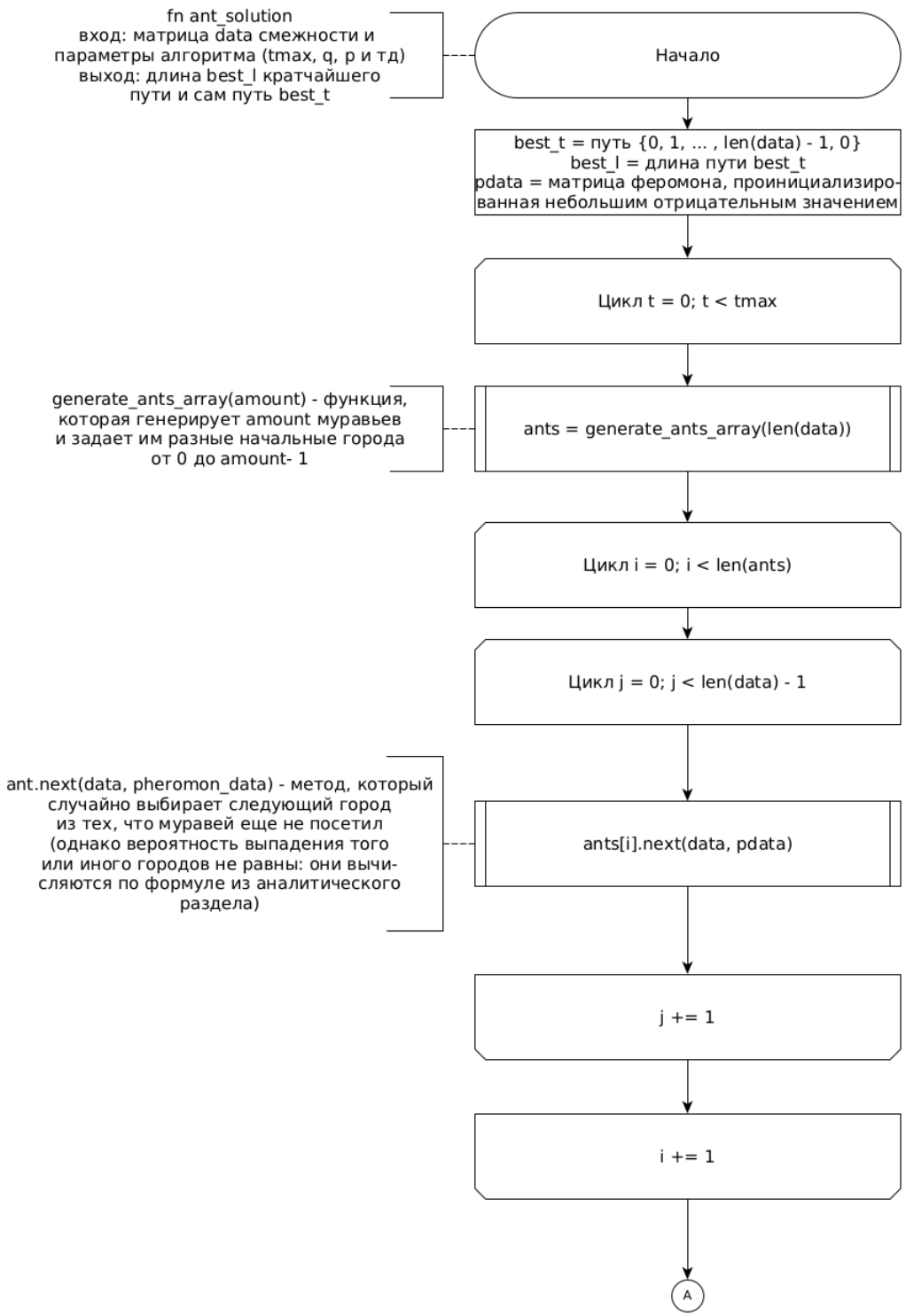


Рис. 2.3: Схема муравьиного алгоритма.

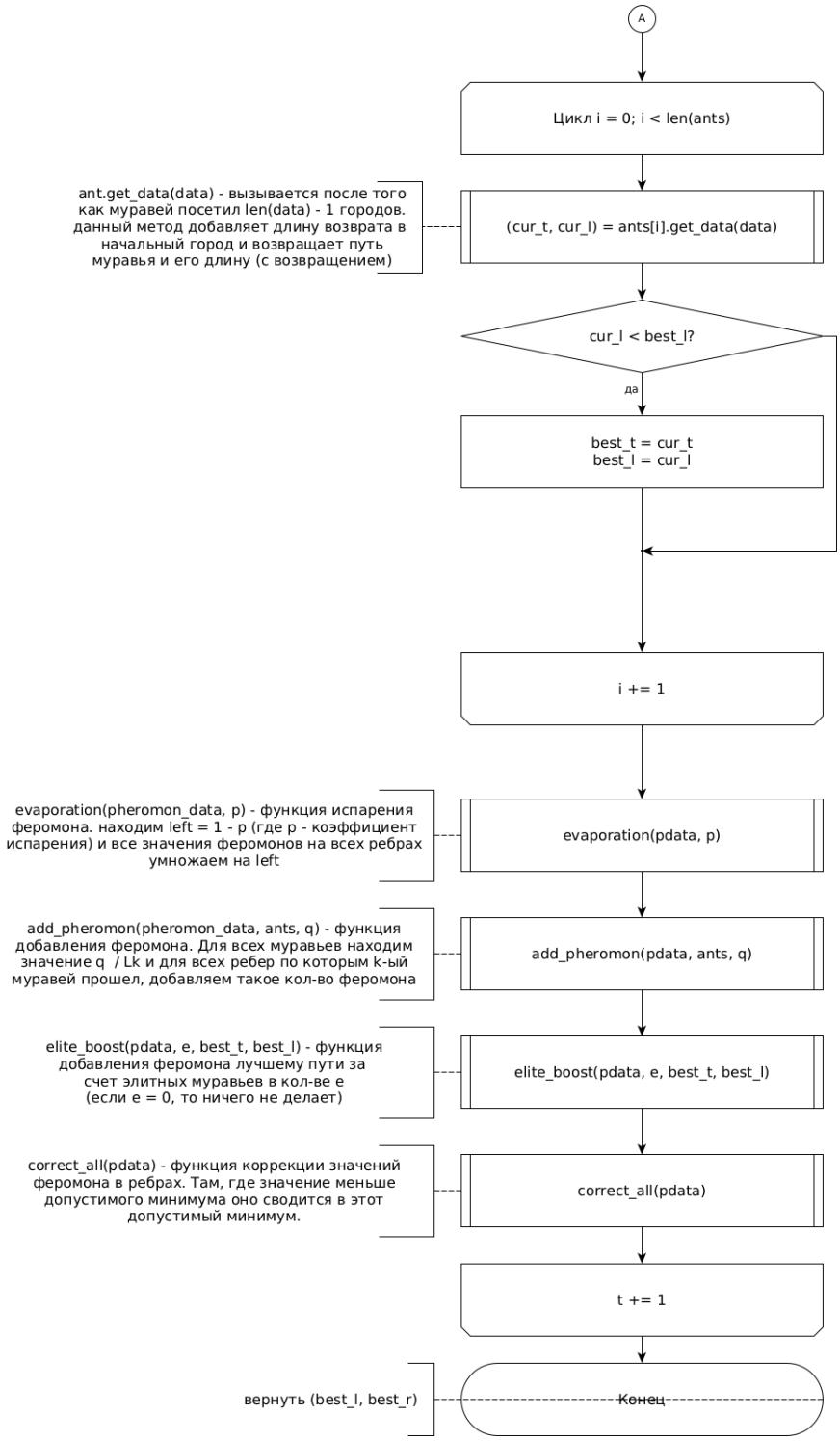


Рис. 2.4: Схема муравьиного алгоритма. Продолжение.

# Вывод

Были представлены схемы алгоритмов полного перебора и муравьиного для решения задачи коммивояжера. Также дополнительно была показана схема алгоритма поиска всех перестановок.

# 3 Технологическая часть

В данном разделе приведены средства программной реализации и листинг кода.

## 3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаётся матрица смежности графа;
- на выходе программа выдаёт минимальную длину и путь, на котором данная длина достигнута.

## 3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран современный компилируемый ЯП Rust [4]. Данный выбор обусловлен популярностью языка и скоростью его выполнения, а также тем, что данный язык предоставляет широкие возможности для написания тестов [5].

## 3.3 Листинги кода

В листинге 3.1 представлена реализация алгоритма полного перебора, в листинге 3.2 представлена реализация муравьиного алгоритма. В листингах 3.3 и 3.4 представлены конфигурационная структура и структура муравья с методами соотвественно.

```
1 use super::Cost;
2 use itertools::Itertools;
3
4 pub struct BruteSolver<'a> {
5     data: &'a [Vec<Cost>],
6 }
7
```

```rust
impl<'a> BruteSolver<'a> {
    pub fn new(data: &'a [Vec<Cost>]) -> Self {
        Self { data }
    }

    pub fn solve(&self) -> (Cost, Vec<usize>) {
        let (mut best_l, mut best_t) = (Cost::MAX, Vec::new());
        for permutation in (1..self.data.len()).permutations(self.data.len() - 1) {
            let l = self.compute_dist(&permutation);
            if l < best_l {
                best_l = l;
                best_t = permutation;
            }
        }
        best_t.insert(0, 0);
        best_t.push(0);
        (best_l, best_t)
    }

    fn compute_dist(&self, t: &[usize]) -> Cost {
        match t.len() {
            0 => 0 as Cost,
            1 => 2 as Cost * self.data[0][t[0]],
            _ => {
                self.data[0][t[0]]
                    + t.windows(2)
                        .fold(0 as Cost, |acc, el| acc + self.data[el[0]][el[1]])
                    + self.data[t[t.len() - 1]][0]
            }
        }
    }
}
```

Листинг 3.1: Реализация полного перебора.

```rust
use super::Cost;
use rand::prelude::*;

mod config;
pub use config::Config;

mod ant;
use ant::Ant;

pub struct AntSolver<'a> {
    data: &'a [Vec<Cost>],
    ndata: Vec<Vec<f64>>,
    q: f64,
    config: Config,
```

```rust
}

impl<'a> AntSolver<'a> {
    pub fn new(data: &'a [Vec<Cost>], config: Config) -> Self {
        let ndata = data
            .iter()
            .map(|row| row.iter().map(|&e| 1_f64 / e as f64).collect())
            .collect();
        let q = Self::compute_q(data);
        Self {
            data,
            ndata,
            q,
            config,
        }
    }

    fn compute_q(data: &[Vec<Cost>]) -> f64 {
        // sum of all
        let q = data
            .iter()
            .fold(0 as Cost, |acc, row| acc + row.iter().sum::<Cost>()) as f64;
        q / data.len() as f64
    }

    pub fn solve(&self) -> (Cost, Vec<usize>) {
        let (mut rng, mut pheromon_data, mut best_t, mut best_l) = self.init_params();
        for _ in 0..self.config.tmax {
            let mut ants = self.generate_ants(&mut rng);
            // Run ants
            for a in ants.iter_mut() {
                a.walk(
                    self.data,
                    &self.ndata,
                    &pheromon_data,
                    self.config.alpha,
                    self.config.beta,
                    &mut rng,
                );
            }

            // Find best T* and L* after day
            let best_data = ants
                .iter()
                .min_by(|a, b| a.data().0.cmp(&b.data().0))
                .unwrap()
                .data();
            if best_data.0 < best_l {
```

```
                best_l = best_data.0;
                best_t = best_data.1.to_vec();
            }

            // Update pheromon: evaporation, add_pheromon by ants,
            // correct_pheromon (min bound), elite_boost (add e * dt to best edges)
            self.evaporation(&mut pheromon_data);
            self.add_pheromon(&ants, &mut pheromon_data);
            self.correct_pheromon(&mut pheromon_data);
            self.elite_boost(&mut pheromon_data, &best_t, best_l);
        }

        best_t.push(best_t[0]);
        (best_l, best_t)
    }

    fn init_params(&self) -> (ThreadRng, Vec<Vec<f64>>, Vec<usize>, Cost) {
        let rng = thread_rng();
        let pheromon_data =
            vec![vec![self.config.pheromon_start; self.data.len()]; self.data.len()];
        let best_t = (0..self.data.len()).collect::<Vec<usize>>();
        let best_l = best_t
            .windows(2)
            .fold(0 as Cost, |acc, win| acc + self.data[win[0]][win[1]])
            + self.data[self.data.len() - 1][0];
        (rng, pheromon_data, best_t, best_l)
    }

    fn generate_ants(&self, rng: &mut ThreadRng) -> Vec<Ant> {
        let (m, data_len) = (self.config.m, self.data.len());
        // placing ants in all places works worse
        let ants = (0..m)
            .map(|_| Ant::new(data_len, rng.gen_range(0, data_len)))
            .collect::<Vec<Ant>>();
        ants
    }

    fn add_pheromon(&self, ants: &[Ant], pdata: &mut [Vec<f64>]) {
        let q = self.q;
        ants.iter().for_each(|ant| {
            let (l, route) = ant.data();
            let val = q / l as f64;
            for path in route.windows(2) {
                pdata[path[0]][path[1]] += val;
                pdata[path[1]][path[0]] += val;
            }
        })
    }
```

```
111
112     fn correct_pheromon(&self, pdata: &mut [Vec<f64>]) {
113         let low_bound = self.config.pheromon_min;
114         pdata.iter_mut().for_each(|row| {
115             row.iter_mut().for_each(|v| {
116                 if *v < low_bound {
117                     *v = low_bound
118                 }
119             })
120         });
121     }
122
123     fn evaporation(&self, pdata: &mut [Vec<f64>]) {
124         let left = 1.0 - self.config.p;
125         pdata
126             .iter_mut()
127             .for_each(|row| row.iter_mut().for_each(|v| *v *= left))
128     }
129
130     fn elite_boost(&self, pdata: &mut [Vec<f64>], best_t: &[usize], best_l: Cost) {
131         let val = self.config.e as f64 * self.q / best_l as f64;
132         best_t.windows(2).for_each(|win| {
133             pdata[win[0]][win[1]] += val;
134             pdata[win[1]][win[0]] += val;
135         });
136         let (first, last) = (best_t[0], best_t[best_t.len() - 1]);
137         pdata[first][last] += val;
138         pdata[last][first] += val;
139     }
140 }
```

Листинг 3.2: Реализация муравьиного алгоритма.

```
1  use serde_derive::Deserialize;
2
3  #[derive(Clone, Deserialize)]
4  pub struct Config {
5      pub alpha: f64,
6      pub beta: f64,
7      pub e: usize, // number of elite ants
8      pub p: f64, // evaporation rate \in [0..1]
9      pub m: usize, // number of ants
10     pub tmax: usize,
11     pub pheromon_start: f64,
12     pub pheromon_min: f64,
13 }
```

Листинг 3.3: Конфигурационная структура.

```rust
use super::Cost;
use rand::prelude::*;

#[derive(Clone)]
pub struct Ant {
    pub route: Vec<usize>,
    len: Cost,
    left: Vec<usize>,
}

impl Ant {
    pub fn new(cities_amount: usize, start: usize) -> Self {
        let left: Vec<usize> = (0..cities_amount).filter(|&e| e != start).collect();
        Self {
            route: vec![start],
            len: 0 as Cost,
            left,
        }
    }

    pub fn walk(
        &mut self,
        d: &[Vec<Cost>],
        nd: &[Vec<f64>],
        pd: &[Vec<f64>],
        alpha: f64,
        beta: f64,
        rng: &mut ThreadRng,
    ) {
        for _ in 0..self.left.len() {
            self.next(d, nd, pd, alpha, beta, rng);
        }
        let last_visited = self.route[self.route.len() - 1];
        self.len += d[last_visited][self.route[0]];
    }

    pub fn data(&self) -> (Cost, &[usize]) {
        (self.len, &self.route)
    }

    fn next(
        &mut self,
        d: &[Vec<Cost>],
        nd: &[Vec<f64>],
        pd: &[Vec<f64>],
        alpha: f64,
        beta: f64,
        rng: &mut ThreadRng,
```

```
    ) {
        let cur = self.route[self.route.len() - 1];
        let denominator = self.left.iter().fold(0.0, |acc, &e| {
            acc + f64::powf(pd[cur][e], alpha) * f64::powf(nd[cur][e], beta)
        });
        let mut pick: f64 = rng.gen();
        for (index, &j) in self.left.iter().enumerate() {
            let cur_prob = f64::powf(pd[cur][j], alpha) * f64::powf(nd[cur][j], beta);
            pick -= cur_prob / denominator;
            if pick < 0_f64 {
                self.pick(index, d[cur][j]);
                return;
            }
        }
        let index = self.left.len() - 1;
        self.pick(index, d[cur][self.left[index]]);
    }

    fn pick(&mut self, index: usize, diff: Cost) {
        self.route.push(self.left.remove(index));
        self.len += diff;
    }
}
```

Листинг 3.4: Структура муравья и её методы.

## 3.4 Тестирование функций.

В таблице 3.1 приведены тесты для функции, реализующей алгоритм для решения задачи коммивояжера. Тесты пройдены успешно.

# Вывод

Спроектированные алгоритмы были реализованы и протестированы.

Таблица 3.1: Тестирование функций

| Матрица смежности | Ожидаемый наименьший путь |
|---|---|
| $\begin{pmatrix} 0 & 9 & 12 & 21 \\ 9 & 0 & 9 & 21 \\ 12 & 9 & 0 & 21 \\ 21 & 21 & 21 & 0 \end{pmatrix}$ | 60, [0, 1, 2, 3, 0] |
| $\begin{pmatrix} 0 & 21 & 12 & 9 \\ 21 & 0 & 12 & 15 \\ 12 & 12 & 0 & 15 \\ 9 & 15 & 15 & 0 \end{pmatrix}$ | 48, [0, 2, 1, 3, 0] |
| $\begin{pmatrix} 0 & 24 & 27 & 21 \\ 24 & 0 & 12 & 12 \\ 27 & 12 & 0 & 6 \\ 21 & 12 & 7 & 0 \end{pmatrix}$ | 63, [0, 1, 2, 3, 0] |

# 4 Исследовательская часть

В данном разделе приведены примеры работы программы и анализ характериситик разработанного программного обеспечения.

## 4.1 Технические характеристики

- Операционная система: Manjaro [6] Linux [7] x86_64.
- Память: 8 ГБ.
- Процессор: Intel® Core™ i7-8550U[8].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

## 4.2 Пример работы программы

На рисунке 4.1 приведен пример работы программы.

```
Data:
    0     2     3     8     4     6     2     5
    2     0     4    11     9     1     3     2
    3     4     0     2     5    11     4     1
    8    11     2     0     4     5    10     2
    4     9     5     4     0     8     3    10
    6     1    11     5     8     0    12     1
    2     3     4    10     3    12     0     9
    5     2     1     2    10     1     9     0


BRUTE:
Length:    16, Path: [0, 1, 5, 7, 2, 3, 4, 6, 0]
ANTS:
Length:    16, Path: [4, 6, 0, 1, 5, 7, 2, 3, 4]
```

Рис. 4.1: Пример работы программы.

## 4.3 Исследование скорости работы алгоритмов.

Алгоритмы тестировались на данных, сгенерированных случайным образом один раз.

Результаты замеров времени приведены в таблице 4.1. На рисунке 4.2 приведен графики зависимостей времени работы алгоритмов от размерности матрицы смежности.

| Размерность матрицы | Полный перебор | Муравьиный алгоритм |
|---|---|---|
| 3 | 1612 | 89996 |
| 4 | 1441 | 151542 |
| 5 | 2319 | 286035 |
| 6 | 7725 | 469782 |
| 7 | 43539 | 710305 |
| 8 | 305564 | 1039688 |
| 9 | 2500918 | 1470419 |
| 10 | 22690076 | 1886565 |

Таблица 4.1: Замеры времени

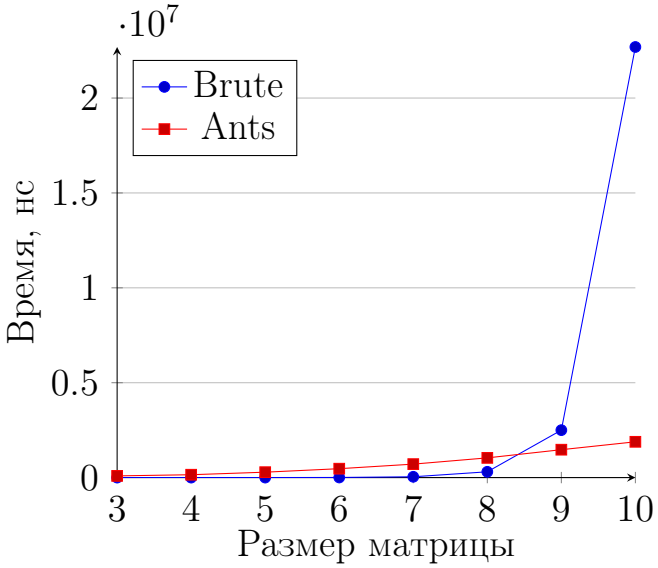


Рис. 4.2: График времени работы алгоритмов.

## 4.4 Постановка эксперимента

В муравьином алгоритме вычисления производятся на основе настраиваемых параметров. Рассмотрим два класса данных и подберем к ним параметры, при которых метод даст точный результат при небольшом количестве итераций.

Будем рассматривать матрицы размерности $10 \times 10$, так как иначе получение точного результата алгоритмом велико и менее зависимо от параметров.

В качестве первого класса данных выделим матрицу смежности, в которой все значения незначительно отличаются друг от друга, например, в диапазоне $[1, 25]$. Вторым классом будут матрицы, где значения могут значительно отличаться, например $[1, 2500]$.

Будем запускать муравьиный алгоритм для всех значений $\alpha, \rho \in [0, 1]$, с шагом $= 0.1$, пока не будет найдено точное значение для каждого набора.

В результате тестирования будет выведена таблица со значениями $\alpha, \beta, \rho, , ,$ где  — оптимальная длина пути, найденная муравьиным алгоритмом,  - Разность между полученным значением и настоящей оптимальной длиной пути, а $\alpha, \beta, \rho$ — настраиваемые параметры.

### 4.4.1 Код тестирования

В листинге 4.1 представлен код, с помощью которого проводилась параметризация.

```
pub fn run_parametrization() {
    let mut conf = read_config(constants::CONFIG_FILE);
    conf.m = constants::TEST_SIZE;
    for (from, to) in [
        (constants::VALS_FROM, constants::VALS_TO),
        (constants::BIG_VALS_FROM, constants::BIG_VALS_TO),
    ]
    .iter()
    {
        let data = generate_data(constants::TEST_SIZE, *from, *to);
        print_data(&data);
        let brute = BruteSolver::new(&data);
        let (best_l, _) = brute.solve();
```

```
14          println!("Best␣Distance:␣{}\n", best_l);
15          conf.alpha = constants::ALPHA_START;
16          while conf.alpha < constants::ALPHA_END + constants::ALPHA_STEP / 2_f64 {
17              conf.beta =
18                  f64::round((1.0 - conf.alpha) * constants::PRESISION) /
                        constants::PRESISION;
19              conf.p = constants::P_START;
20              while conf.p < constants::P_END + constants::P_STEP / 2_f64 {
21                  let ant_solver = AntSolver::new(&data, conf.clone());
22                  let (len, _) = ant_solver.solve();
23                  println!(
24                      "{:<4}␣&␣{:<4}␣&␣{:<4}␣&␣{:<5}␣&␣{:<5}␣\\\\",
25                      conf.alpha,
26                      conf.beta,
27                      conf.p,
28                      len,
29                      len - best_l
30                  );
31                  conf.p = f64::round((conf.p + constants::P_STEP) * constants::PRESISION)
32                      / constants::PRESISION;
33              }
34              conf.alpha = f64::round((conf.alpha + constants::ALPHA_STEP) *
                    constants::PRESISION)
35                  / constants::PRESISION;
36          }
37      }
38  }
39
40  fn print_data(data: &[Vec<Cost>]) {
41      data.iter().for_each(|row| {
42          print!("{:>5}␣", row[0]);
43          row.iter().skip(1).for_each(|e| print!("&␣{:>5}␣", e));
44          println!("\\\\");
45      })
46  }
```

Листинг 4.1: Код для параметризации.

### 4.4.2 Класс данных 1

$$M = \begin{pmatrix} 0 & 21 & 21 & 4 & 22 & 20 & 6 & 4 & 11 & 23 \\ 21 & 0 & 22 & 6 & 9 & 21 & 16 & 6 & 2 & 7 \\ 21 & 22 & 0 & 18 & 23 & 7 & 19 & 19 & 1 & 14 \\ 4 & 6 & 18 & 0 & 7 & 11 & 16 & 10 & 8 & 19 \\ 22 & 9 & 23 & 7 & 0 & 15 & 2 & 23 & 1 & 22 \\ 20 & 21 & 7 & 11 & 15 & 0 & 6 & 7 & 8 & 21 \\ 6 & 16 & 19 & 16 & 2 & 6 & 0 & 8 & 3 & 14 \\ 4 & 6 & 19 & 10 & 23 & 7 & 8 & 0 & 23 & 15 \\ 11 & 2 & 1 & 8 & 1 & 8 & 3 & 23 & 0 & 9 \\ 23 & 7 & 14 & 19 & 22 & 21 & 14 & 15 & 9 & 0 \end{pmatrix} \tag{4.1}$$

В таблице 4.2 приведены результаты параметризации метода решения задачи коммивояжера на основании муравьиного алгоритма. Количество дней было взято равным 50. Полный перебор определил оптимальную длину пути 52. Два последних стобца таблицы определяют найденный муравьиным алгоритм оптимальный путь и разницу этого пути с оптимальным путём, найденным алгоритмом полного перебора.

Таблица 4.2: Таблица коэффициентов для класса данных 1.

| $\alpha$ | $\beta$ | $\rho$ | Длина | Разница |
|---|---|---|---|---|
| 0 | 1 | 0 | 52 | 0 |
| 0 | 1 | 0.1 | 52 | 0 |
| 0 | 1 | 0.2 | 52 | 0 |
| 0 | 1 | 0.3 | 52 | 0 |
| 0 | 1 | 0.4 | 53 | 1 |
| 0 | 1 | 0.5 | 52 | 0 |
| 0 | 1 | 0.6 | 52 | 0 |
| 0 | 1 | 0.7 | 52 | 0 |
| 0 | 1 | 0.8 | 52 | 0 |
| 0 | 1 | 0.9 | 52 | 0 |
| 0 | 1 | 1 | 52 | 0 |
| 0.1 | 0.9 | 0 | 52 | 0 |
| 0.1 | 0.9 | 0.1 | 52 | 0 |
| 0.1 | 0.9 | 0.2 | 53 | 1 |
| 0.1 | 0.9 | 0.3 | 52 | 0 |
| 0.1 | 0.9 | 0.4 | 52 | 0 |
| 0.1 | 0.9 | 0.5 | 52 | 0 |
| 0.1 | 0.9 | 0.6 | 53 | 1 |
| 0.1 | 0.9 | 0.7 | 52 | 0 |
| 0.1 | 0.9 | 0.8 | 52 | 0 |
| 0.1 | 0.9 | 0.9 | 52 | 0 |
| 0.1 | 0.9 | 1 | 52 | 0 |
| 0.2 | 0.8 | 0 | 52 | 0 |
| 0.2 | 0.8 | 0.1 | 52 | 0 |
| 0.2 | 0.8 | 0.2 | 52 | 0 |
| 0.2 | 0.8 | 0.3 | 53 | 1 |
| 0.2 | 0.8 | 0.4 | 52 | 0 |
| 0.2 | 0.8 | 0.5 | 52 | 0 |
| 0.2 | 0.8 | 0.6 | 53 | 1 |
| 0.2 | 0.8 | 0.7 | 53 | 1 |
| 0.2 | 0.8 | 0.8 | 52 | 0 |
| 0.2 | 0.8 | 0.9 | 52 | 0 |
| 0.2 | 0.8 | 1 | 52 | 0 |
| 0.3 | 0.7 | 0 | 53 | 1 |
| 0.3 | 0.7 | 0.1 | 52 | 0 |
| 0.3 | 0.7 | 0.2 | 52 | 0 |
| 0.3 | 0.7 | 0.3 | 53 | 1 |
| 0.3 | 0.7 | 0.4 | 52 | 0 |
| 0.3 | 0.7 | 0.5 | 52 | 0 |
| 0.3 | 0.7 | 0.6 | 52 | 0 |
| 0.3 | 0.7 | 0.7 | 52 | 0 |
| 0.3 | 0.7 | 0.8 | 52 | 0 |
| 0.3 | 0.7 | 0.9 | 53 | 1 |
| 0.3 | 0.7 | 1 | 52 | 0 |
| 0.4 | 0.6 | 0 | 53 | 1 |
| 0.4 | 0.6 | 0.1 | 53 | 1 |
| 0.4 | 0.6 | 0.2 | 53 | 1 |
| 0.4 | 0.6 | 0.3 | 52 | 0 |
| 0.4 | 0.6 | 0.4 | 53 | 1 |
| 0.4 | 0.6 | 0.5 | 52 | 0 |
| 0.4 | 0.6 | 0.6 | 52 | 0 |
| 0.4 | 0.6 | 0.7 | 52 | 0 |
| 0.4 | 0.6 | 0.8 | 52 | 0 |
| 0.4 | 0.6 | 0.9 | 52 | 0 |
| 0.4 | 0.6 | 1 | 52 | 0 |
| 0.5 | 0.5 | 0 | 52 | 0 |
| 0.5 | 0.5 | 0.1 | 52 | 0 |
| 0.5 | 0.5 | 0.2 | 52 | 0 |
| 0.5 | 0.5 | 0.3 | 53 | 1 |
| 0.5 | 0.5 | 0.4 | 52 | 0 |
| 0.5 | 0.5 | 0.5 | 52 | 0 |
| 0.5 | 0.5 | 0.6 | 52 | 0 |
| 0.5 | 0.5 | 0.7 | 52 | 0 |
| 0.5 | 0.5 | 0.8 | 52 | 0 |
| 0.5 | 0.5 | 0.9 | 52 | 0 |
| 0.5 | 0.5 | 1 | 52 | 0 |
| 0.6 | 0.4 | 0 | 53 | 1 |
| 0.6 | 0.4 | 0.1 | 53 | 1 |
| 0.6 | 0.4 | 0.2 | 56 | 4 |
| 0.6 | 0.4 | 0.3 | 52 | 0 |
| 0.6 | 0.4 | 0.4 | 52 | 0 |
| 0.6 | 0.4 | 0.5 | 55 | 3 |
| 0.6 | 0.4 | 0.6 | 56 | 4 |
| 0.6 | 0.4 | 0.7 | 52 | 0 |
| 0.6 | 0.4 | 0.8 | 53 | 1 |
| 0.6 | 0.4 | 0.9 | 53 | 1 |
| 0.6 | 0.4 | 1 | 52 | 0 |
| 0.7 | 0.3 | 0 | 52 | 0 |
| 0.7 | 0.3 | 0.1 | 53 | 1 |
| 0.7 | 0.3 | 0.2 | 52 | 0 |

| $\alpha$ | $\beta$ | $\rho$ | Длина | Разница |
|---|---|---|---|---|
| 0.7 | 0.3 | 0.3 | 52 | 0 |
| 0.7 | 0.3 | 0.4 | 53 | 1 |
| 0.7 | 0.3 | 0.5 | 53 | 1 |
| 0.7 | 0.3 | 0.6 | 52 | 0 |
| 0.7 | 0.3 | 0.7 | 53 | 1 |
| 0.7 | 0.3 | 0.8 | 57 | 5 |
| 0.7 | 0.3 | 0.9 | 52 | 0 |
| 0.7 | 0.3 | 1 | 52 | 0 |
| 0.8 | 0.2 | 0 | 59 | 7 |
| 0.8 | 0.2 | 0.1 | 53 | 1 |
| 0.8 | 0.2 | 0.2 | 56 | 4 |
| 0.8 | 0.2 | 0.3 | 53 | 1 |
| 0.8 | 0.2 | 0.4 | 52 | 0 |
| 0.8 | 0.2 | 0.5 | 56 | 4 |
| 0.8 | 0.2 | 0.6 | 53 | 1 |
| 0.8 | 0.2 | 0.7 | 52 | 0 |
| 0.8 | 0.2 | 0.8 | 52 | 0 |
| 0.8 | 0.2 | 0.9 | 52 | 0 |
| 0.8 | 0.2 | 1 | 53 | 1 |
| 0.9 | 0.1 | 0 | 56 | 4 |
| 0.9 | 0.1 | 0.1 | 53 | 1 |
| 0.9 | 0.1 | 0.2 | 52 | 0 |
| 0.9 | 0.1 | 0.3 | 56 | 4 |
| 0.9 | 0.1 | 0.4 | 52 | 0 |
| 0.9 | 0.1 | 0.5 | 53 | 1 |
| 0.9 | 0.1 | 0.6 | 56 | 4 |
| 0.9 | 0.1 | 0.7 | 56 | 4 |
| 0.9 | 0.1 | 0.8 | 55 | 3 |
| 0.9 | 0.1 | 0.9 | 53 | 1 |
| 0.9 | 0.1 | 1 | 53 | 1 |
| 1 | 0 | 0 | 71 | 19 |
| 1 | 0 | 0.1 | 61 | 9 |
| 1 | 0 | 0.2 | 53 | 1 |
| 1 | 0 | 0.3 | 59 | 7 |
| 1 | 0 | 0.4 | 59 | 7 |
| 1 | 0 | 0.5 | 60 | 8 |
| 1 | 0 | 0.6 | 60 | 8 |
| 1 | 0 | 0.7 | 74 | 22 |
| 1 | 0 | 0.8 | 60 | 8 |
| 1 | 0 | 0.9 | 57 | 5 |
| 1 | 0 | 1 | 60 | 8 |

### 4.4.3 Класс данных 2

$$M = \begin{pmatrix} 0 & 1790 & 200 & 1900 & 63 & 1659 & 1820 & 1395 & 2382 & 649 \\ 1790 & 0 & 1573 & 2435 & 1515 & 714 & 892 & 2193 & 1590 & 1003 \\ 200 & 1573 & 0 & 833 & 392 & 2404 & 962 & 902 & 141 & 1123 \\ 1900 & 2435 & 833 & 0 & 2283 & 1652 & 2362 & 2262 & 1512 & 2166 \\ 63 & 1515 & 392 & 2283 & 0 & 1322 & 290 & 1305 & 2100 & 969 \\ 1659 & 714 & 2404 & 1652 & 1322 & 0 & 256 & 78 & 2236 & 2041 \\ 1820 & 892 & 962 & 2362 & 290 & 256 & 0 & 1180 & 1547 & 1279 \\ 1395 & 2193 & 902 & 2262 & 1305 & 78 & 1180 & 0 & 1640 & 1161 \\ 2382 & 1590 & 141 & 1512 & 2100 & 2236 & 1547 & 1640 & 0 & 2212 \\ 649 & 1003 & 1123 & 2166 & 969 & 2041 & 1279 & 1161 & 2212 & 0 \end{pmatrix} \tag{4.2}$$

В таблице 4.3 приведены результаты параметризации метода решения задачи коммивояжера на основании муравьиного алгоритма для матрицы с элементами в диапазоне $[0, 2500]$. Количество дней было взято равным 50. Полный перебор определил оптимальную длину пути 6986. Два последних стобца таблицы определяют найденный муравьиным алгоритм оптимальный путь и разницу этого пути с оптимальным путём, найденным алгоритмом полного перебора.

Таблица 4.3: Таблица коэффициентов для класса данных №2

| $\alpha$ | $\beta$ | $\rho$ | Длина | Разница |
|---|---|---|---|---|
| 0 | 1 | 0 | 6986 | 0 |
| 0 | 1 | 0.1 | 6986 | 0 |
| 0 | 1 | 0.2 | 6986 | 0 |
| 0 | 1 | 0.3 | 6986 | 0 |
| 0 | 1 | 0.4 | 6986 | 0 |
| 0 | 1 | 0.5 | 6986 | 0 |
| 0 | 1 | 0.6 | 6986 | 0 |
| 0 | 1 | 0.7 | 6986 | 0 |
| 0 | 1 | 0.8 | 6986 | 0 |
| 0 | 1 | 0.9 | 6992 | 6 |
| 0 | 1 | 1 | 6986 | 0 |
| 0.1 | 0.9 | 0 | 6986 | 0 |
| 0.1 | 0.9 | 0.1 | 6992 | 6 |
| 0.1 | 0.9 | 0.2 | 6986 | 0 |
| 0.1 | 0.9 | 0.3 | 6986 | 0 |
| 0.1 | 0.9 | 0.4 | 6986 | 0 |
| 0.1 | 0.9 | 0.5 | 6986 | 0 |
| 0.1 | 0.9 | 0.6 | 6986 | 0 |
| 0.1 | 0.9 | 0.7 | 6986 | 0 |
| 0.1 | 0.9 | 0.8 | 6986 | 0 |
| 0.1 | 0.9 | 0.9 | 7165 | 179 |
| 0.1 | 0.9 | 1 | 6986 | 0 |
| 0.2 | 0.8 | 0 | 6986 | 0 |
| 0.2 | 0.8 | 0.1 | 6986 | 0 |
| 0.2 | 0.8 | 0.2 | 6986 | 0 |
| 0.2 | 0.8 | 0.3 | 6992 | 6 |
| 0.2 | 0.8 | 0.4 | 6992 | 6 |
| 0.2 | 0.8 | 0.5 | 6992 | 6 |
| 0.2 | 0.8 | 0.6 | 6986 | 0 |
| 0.2 | 0.8 | 0.7 | 6992 | 6 |
| 0.2 | 0.8 | 0.8 | 6986 | 0 |
| 0.2 | 0.8 | 0.9 | 6986 | 0 |
| 0.2 | 0.8 | 1 | 6986 | 0 |
| 0.3 | 0.7 | 0 | 6986 | 0 |
| 0.3 | 0.7 | 0.1 | 6986 | 0 |
| 0.3 | 0.7 | 0.2 | 7139 | 153 |
| 0.3 | 0.7 | 0.3 | 7139 | 153 |
| 0.3 | 0.7 | 0.4 | 6986 | 0 |
| 0.3 | 0.7 | 0.5 | 6986 | 0 |
| 0.3 | 0.7 | 0.6 | 6986 | 0 |
| 0.3 | 0.7 | 0.7 | 6986 | 0 |
| 0.3 | 0.7 | 0.8 | 6992 | 6 |
| 0.3 | 0.7 | 0.9 | 6992 | 6 |
| 0.3 | 0.7 | 1 | 6986 | 0 |
| 0.4 | 0.6 | 0 | 6986 | 0 |
| 0.4 | 0.6 | 0.1 | 6992 | 6 |
| 0.4 | 0.6 | 0.2 | 6986 | 0 |
| 0.4 | 0.6 | 0.3 | 6986 | 0 |
| 0.4 | 0.6 | 0.4 | 6986 | 0 |
| 0.4 | 0.6 | 0.5 | 6992 | 6 |
| 0.4 | 0.6 | 0.6 | 6992 | 6 |
| 0.4 | 0.6 | 0.7 | 6986 | 0 |
| 0.4 | 0.6 | 0.8 | 7139 | 153 |
| 0.4 | 0.6 | 0.9 | 6986 | 0 |
| 0.4 | 0.6 | 1 | 6992 | 6 |
| 0.5 | 0.5 | 0 | 7139 | 153 |
| 0.5 | 0.5 | 0.1 | 6986 | 0 |
| 0.5 | 0.5 | 0.2 | 6986 | 0 |
| 0.5 | 0.5 | 0.3 | 7139 | 153 |
| 0.5 | 0.5 | 0.4 | 6986 | 0 |
| 0.5 | 0.5 | 0.5 | 6986 | 0 |
| 0.5 | 0.5 | 0.6 | 6986 | 0 |
| 0.5 | 0.5 | 0.7 | 6986 | 0 |
| 0.5 | 0.5 | 0.8 | 6986 | 0 |
| 0.5 | 0.5 | 0.9 | 6986 | 0 |
| 0.5 | 0.5 | 1 | 6986 | 0 |
| 0.6 | 0.4 | 0 | 7139 | 153 |
| 0.6 | 0.4 | 0.1 | 6992 | 6 |
| 0.6 | 0.4 | 0.2 | 6986 | 0 |
| 0.6 | 0.4 | 0.3 | 6986 | 0 |
| 0.6 | 0.4 | 0.4 | 7139 | 153 |
| 0.6 | 0.4 | 0.5 | 6992 | 6 |
| 0.6 | 0.4 | 0.6 | 6986 | 0 |
| 0.6 | 0.4 | 0.7 | 6986 | 0 |
| 0.6 | 0.4 | 0.8 | 6986 | 0 |
| 0.6 | 0.4 | 0.9 | 6992 | 6 |
| 0.6 | 0.4 | 1 | 6986 | 0 |
| 0.7 | 0.3 | 0 | 6986 | 0 |
| 0.7 | 0.3 | 0.1 | 6986 | 0 |
| 0.7 | 0.3 | 0.2 | 6986 | 0 |
| 0.7 | 0.3 | 0.3 | 7139 | 153 |

| $\alpha$ | $\beta$ | $\rho$ | Длина | Разница |
|---|---|---|---|---|
| 0.7 | 0.3 | 0.4 | 7165 | 179 |
| 0.7 | 0.3 | 0.5 | 7139 | 153 |
| 0.7 | 0.3 | 0.6 | 6992 | 6 |
| 0.7 | 0.3 | 0.7 | 6992 | 6 |
| 0.7 | 0.3 | 0.8 | 6986 | 0 |
| 0.7 | 0.3 | 0.9 | 6992 | 6 |
| 0.7 | 0.3 | 1 | 6986 | 0 |
| 0.8 | 0.2 | 0 | 7139 | 153 |
| 0.8 | 0.2 | 0.1 | 7562 | 576 |
| 0.8 | 0.2 | 0.2 | 6992 | 6 |
| 0.9 | 0.1 | 0.2 | 6992 | 6 |
| 0.9 | 0.1 | 0.3 | 6986 | 0 |
| 0.9 | 0.1 | 0.4 | 7139 | 153 |
| 0.9 | 0.1 | 0.5 | 7329 | 343 |
| 0.9 | 0.1 | 0.6 | 7217 | 231 |
| 0.9 | 0.1 | 0.7 | 7139 | 153 |
| 0.9 | 0.1 | 0.8 | 7217 | 231 |
| 0.9 | 0.1 | 0.9 | 7376 | 390 |
| 0.9 | 0.1 | 1 | 6986 | 0 |
| 1 | 0 | 0 | 8531 | 1545 |
| 1 | 0 | 0.1 | 8588 | 1602 |
| 1 | 0 | 0.2 | 6986 | 0 |
| 1 | 0 | 0.3 | 7720 | 734 |
| 1 | 0 | 0.4 | 7554 | 568 |
| 1 | 0 | 0.5 | 6992 | 6 |
| 1 | 0 | 0.6 | 7920 | 934 |
| 1 | 0 | 0.7 | 7217 | 231 |
| 1 | 0 | 0.8 | 7874 | 888 |
| 1 | 0 | 0.9 | 7446 | 460 |
| 1 | 0 | 1 | 8119 | 1133 |

# Вывод

В результате сравнения алгоритма полного перебора и муравьиного алгоритма по времени из таблицы 4.1 были получены следующие результаты:

- при относительно небольших размерах матрицы смежности (а именно от 3 до 8) алгоритм полного перебора работает значительно быстрее (при размере 3 — $\approx$ в 50 раз, при размере 6 — $\approx$ в 60 раз);
- при размерах матрицы смежности 9 и выше, время работы алгоритма полного перебора начинает резко возрастать, и становится при размере 9 на 78% медленнее муравьиного алгоритма, а на размере 10 алгоритм полного перебора работает $\approx$ в 12 раз дольше, нежели муравьиный алгоритм.

На основе проведенной параметризации для двух классов данных можно сделать следующие выводы:

- Для класса данных №1, содержащего приблизительно равные значения, наилучшими наборами стали ($\alpha = 0.5, \beta = 0.5, \rho =$ любое), так как они показали наиболее стабильные результаты, равные эталонному значению оптимального пути, равного 52 единицам.
- Для класса данных №2, содержащего различные значения, наилучшими наборами стали ($\alpha = 0.5, \beta = 0.5, \rho =$ любое). При этих параметрах, количество найденных эталонных оптимальных путей составило 8 единиц.

# Заключение

В рамках лабораторной работы были выполнены следующие задачи:

- был изучен и реализован алгоритм полного перебора для решения задачи коммивояжера;
- был изучен и реализован муравьиный алгоритм для решения задачи коммивояжера;
- была проведена параметризация муравьиного алгоритма на двух классах данных;
- был проведён сравнительный анализ скорости работы реализованных алгоритмов;
- был сделан отчёт по проведенной работе.

Исследования показали, что муравьиный алгоритм решения задачи коммивояжера преобладает по скорости более чем в 12 раз над алгоритмом полного перебора в случае с графами с количеством вершин большим или равным 10. Однако, в отличии от алгоритма полного перебора, муравьиный алгоритм не гарантирует, что найденный путь будет оптимальным, так как является эвристическим алгоритмом.

# Литература


[1] Задача коммивояжёра. Режим доступа: http://mech.math.msu.su/~shvetz/54/inf/perl-problems/chCommisVoyageur.xhtml (дата обращения 09.12.2020).

[2] Introduction to Complexity Theory Column 36. Режим доступа: http://www.cs.umd.edu/~gasarch/papers/poll.pdf (дата обращения: 12.12.2020).

[3] М.В. Ульянов. Ресурсно-эффективные компьютерные алгоритмы. ФИЗМАТЛИТ, 2008. с. 304.

[4] Rust Programming Language [Электронный ресурс]. URL: https://doc.rust-lang.org/std/index.html.

[5] Документация по ЯП Rust: бенчмарки [Электронный ресурс]. Режим доступа: https://doc.rust-lang.org/1.7.0/book/benchmark-tests.html (дата обращения: 10.10.2020).

[6] Manjaro – enjoy the simplicity [Электронный ресурс]. Режим доступа: https://manjaro.org/ (дата обращения: 10.10.2020).

[7] Русская информация об ОС Linux [Электронный ресурс]. Режим доступа: https://www.linux.org.ru/ (дата обращения: 10.10.2020).

[8] Процессор Intel® Core™ i7-8550U [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/122589/intel-core-i7-8550u-processor-8m-cache-up-to-4-00-ghz.html (дата обращения: 10.10.2020).