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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

О Т Ч Е Т

по лабораторной работе № 6

Название: Решение задачи коммивояжёра

Дисциплина: Анализ алгоритмов

Студент	ИУ7-52Б		В.А. Иванов
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2020

Оглавление

Введение

В данной лабораторной реализуются и оцениваются алгоритмы решения задачи коммивояжёра.

Задача коммивояжёра является одним из самых известных примеров NP-полной задачи. Она заключается, в поиске наиболее выгодного маршрута, проходящего однократно через все вершины графа, кроме начальной вершины, которая должна оказаться и конечной.

Интерес к данной задачи обусловлен тем, что на данный момент не существует алгоритма, способного находить её решение за полиномиальное время в зависимости от количества вершин. При этом, известны алгоритмы, которые способны найти маршрут, который по длине будет достаточно приближён к по наилучшему решению. Этот факт позволяет принимать подобные решения в практике, когда "почти идеальное"решение более чем удволетворяет требованиям решаемой проблемы. Примером подобной задачи является поиск маршрута пайки печатной платы, при котором манипулятор-пайщик проделает наименьший путь между контактами. В данном случае возможно использование достаточно короткого, но не лучшего маршрута.

В данной лабораторной работе в качестве алгоритмов поиска решения будут рассмотрены:

- поиск полным перебором;
- поиск муравьиным алгоритмом.

В первом случае будет измерения длины всех возможных маршрутов. Это является достаточно затратным решением, но гарантированно будет получено наилучшее решение.

Второй алгоритм является воплощением механизма, созданого самой природой – поведением колонией муравьёв. Суть поведения каждого муравья заключается в использовании опыта ранее ходивших муравьёв в принятии решения о выборе следующей вершины. Опыт задаётся при помощи откладывания на рёбрах графа феромона, который тем больше, чем оптимальнее маршрут проходящий через данное ребро. Таким образом, спустя множество поколений муравьёв, пользующихся знаниями своих предков, можно выявить наиболее оптимальный вариант прохождения.

1. Аналитическая часть

1.1. Цель и задачи работы

Целью лабораторной работы является проведение сравнительного анализ метода полного перебора и эвристического метода на базе муравьиного алгоритма.

Выделены следующие задачи лабораторной работы:

- описание задачи коммивояжёра;
- описание и реализация метода полного перебора и метода на базе муравьиного алгоритма для решения задачи коммивояжёра;
- оценка трудоёмкости муравьиного алгоритма по результатам эксперементальных замеров времени работы;
- проведение параметризации муравьиного метода (определение параметров, для которых метод даёт наилучшие результаты на выбранных классах задач).

1.2. Описание задачи коммивояжёра

Задача заключается в поиске гамильтонова цикла (т.е. замкнутый путь, проходящий через каждую вершину ровно один раз) на неориентированном графе G, с количеством вершин N[1]. Вес рёбер можно задать с помощью квадратной матрицы D размером N, где D_{ij} равняется стоимости перехода из вершины i в j. Сами маршруты можно представить как массив M длиной $N + 1$, где M_i - вершина, посещённая в i-ю очередь.

1.3. Поиск полным перебором

Данный алгоритм составляет все возможные маршруты, начинающиеся из нулевой вершины, и измеряет длину каждого из вариантов. Маршрут с минимальной длиной гарантированно будет являться решением поставленной задачи.

Каждый маршрут начинается и заканчивается в нулевой вершине, потому что каждый путь обязательно будет содержать эту вершину, а так как это цикл, то любая последовательность посещения вершин может быть преобразована в маршрут из нулевой вершины, обладающий той же длиной. Поэтому, не имеет смысла рассмотрение иных начальных вершин.

1.4. Поиск муравьиным алгоритмом

Алгоритм симулирует поведение N муравьёв, которые вместе называются колонией. Колония существует max_t дней. В начале t-го дня по всем вершинам выставляется по муравью. Каждый муравей совершает попытку построить гамельтонов цикл. В случае удачного построения цикла на пройденых рёбрах им выставляется определённое количество феромона. После этого наступает t-я ночь, в которой часть феромона улетучивается, после чего начинается следующий день. Количество феромона на ребре $i - j$ в момент времени t обозначается как $\tau_{ij}(t)$

После симуляции всех дней алгоритм выдаёт в качестве решения наикратчайший маршрут среди всех пройденых. Стоит оговорить то, далеко не всегда этот маршрут будет являться правильным решением поставленной задачи, так как вероятнее всего, алгоритм проверит все возможные пути в графе.

Рассмотрим принцип формирования маршрута. Оказываясь в очередной вершине i (кроме заключительной), муравей совершает выбор одной из доступных для перехода вершин, которые ещё не были посещены. Выбор основывается на величине, определяемой формулой 1.1

$$P_{ij}(t) = [\tau_{ij}(t)]^{\alpha}/[D_{ij}]^{\beta} \tag{1.1}$$

где α, β - коэффициенты стадности и жадности.

Полученные значения можно пронормировать, поделив их на сумму всех величин, в таком случае получится вероятность перехода в j-ю вершину. Основываясь на этом муравей делает случайный выбор следующей вершины с заданными вероятностями.

Каждый муравей прошедший полный маршрут увеличивает значение феромона в посещённых рёбрах на величину, указанной в формуле 1.2

$$\Delta\tau_{ijk}(t) = Q/L_k(t) \tag{1.2}$$

где k - номер муравья, Q - параметр, по порядку приближенный к ожидаемой минимальной длине пути, $L_k(t)$ - путь пройденый k-м муравьём в день t, $\tau_{ijk}(t)$ - приращение феромона от k-го муравья в день t

Для акцентирования феромонов на лучшем пути также используется EL элитных муравьёв, которые каждый проходят по наикратчайшему маршруту на данный момент и, как и обычные муравьи, оставляют феромоны по формуле1.2

После учёта всех приращений происходит испарение феромона, т.е. значение феромона в следующий день вычисляется как 1.3

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{N} \Delta\tau_{ijk}(t) \tag{1.3}$$

где ρ - коэффициент испарения феромона ($\rho \in [0, 1]$).

Вывод

Результатом аналитического раздела стало определение цели и задач работы, описана задача коммивояжёра и алгоритмы поиска.

2. Конструкторская часть

Рассмотрим описанные алгоритмы поиска маршрута. Пусть поиск производится по квадратной матрице растояний D размером N.

2.1. Поиск полным перебором

Алгоритм начинает в 0-й вершине и передаёт в рекурсивную функцию текущую позицию и список доступных вершин, содержащий все вершины кроме начальной. Далее функция поочерёдно производит выбор каждой из возможных вершин в качестве следующей для перехода, учитывает расстояние от предыдущей вершины и вызывает эту же функцию, исключив из списка доступных вершин выбранную. Функция возвращает наиболее короткий путь из всех найденых вариантов, после чего добавляет в него текущую вершину и возвращает в качестве самого короткого пути. Функция, получившая пустое множество доступных вершин совершает переход в начальную вершину.

Схема алгоритма приведена на рисунке 2.1

2.2. Поиск муравьиным алгоритмом

Изначально создаётся матрица феромонов tau, заполненая небольшим положительным числом, минимальный путь и его длина. После этого начинается цикл по дням от 0 до max_t.

Каждый муравей содержит информаицю о текущей позиции, проделанном пути и доступных для посещения вершинах. В начале дня создаётся массив муравьёв размером N. Каждый из муравьёв помещается в незанятую другим муравьём вершину. Далее производится цикл по каждому из муравьёв.

Для очередного k-го муравья осуществляется оценка доступных вершин и переход в одну из них по случайному выбору с найдеными вероятностями. Данные действия повторяются до исчерпания доступных вершин, после чего совершается переход в начальную вершину. В случае, если муравей попадает в тупик, то он останавливается на месте, а результат его прохождения не учитывается далее. В конце пути каждого муравья обновляется значение минимального маршрута.

После конца цикла по муравьям производится создание матрицы $dtau$ приращения феромонов и её заполнение в соответсвии с пройдеными путями. Также симулируется прохождение элитных муравьёв по текущему лучшему пути. После этого производится испарение феромона и занесение значений из $dtau$ в tau. Конролируется итоговое значение ячеек tau – оно не должно опускаться ниже 0.1 от начальной величины.

Схема алгоритма приведена на рисунках 2.2 и 2.3

2.3. Автоматическая параметризация муравьиного алгоритма

Так как муравьиный алгоритм зависит от множества факторов, оптимальные значения используемых параметров могут сильно отличаться в зависимости от класса исследуемого графа. Для поиска наиболее оптимальных параметров требуется создать алгоритм перебора значений параметров, который будет выдавать результат работы для каждого набора и осуществлять поиск наиболее удачных параметров.

2.4. Требования к программному обеспечению

Для полноценной проверки и оценки алгоритмов необходимо выполнить следующее.

1. Предоставить возможность ввода матрицы расстояний и проверяемого алгоритма.
2. Реализовать функцию профилирования, производящую испытания с различными параметрами α, β, ρ.

2.5. Заготовки тестов

При проверке алгоритма необходимо будет использовать следующие классы тестов:

- поиск при двух вершинах;
- поиск в графе, где невозможен гамильтонов цикл;
- поиск в графе, где все расстояния равны;
- поиск в произвольном графе.

Вывод

Результатом конструторской части стало схематическое описание алгоритмов поиска и параметризации, сформулированны тесты и требования к программному обеспечению.

Рис. 2.1 — Поиск полным перебором

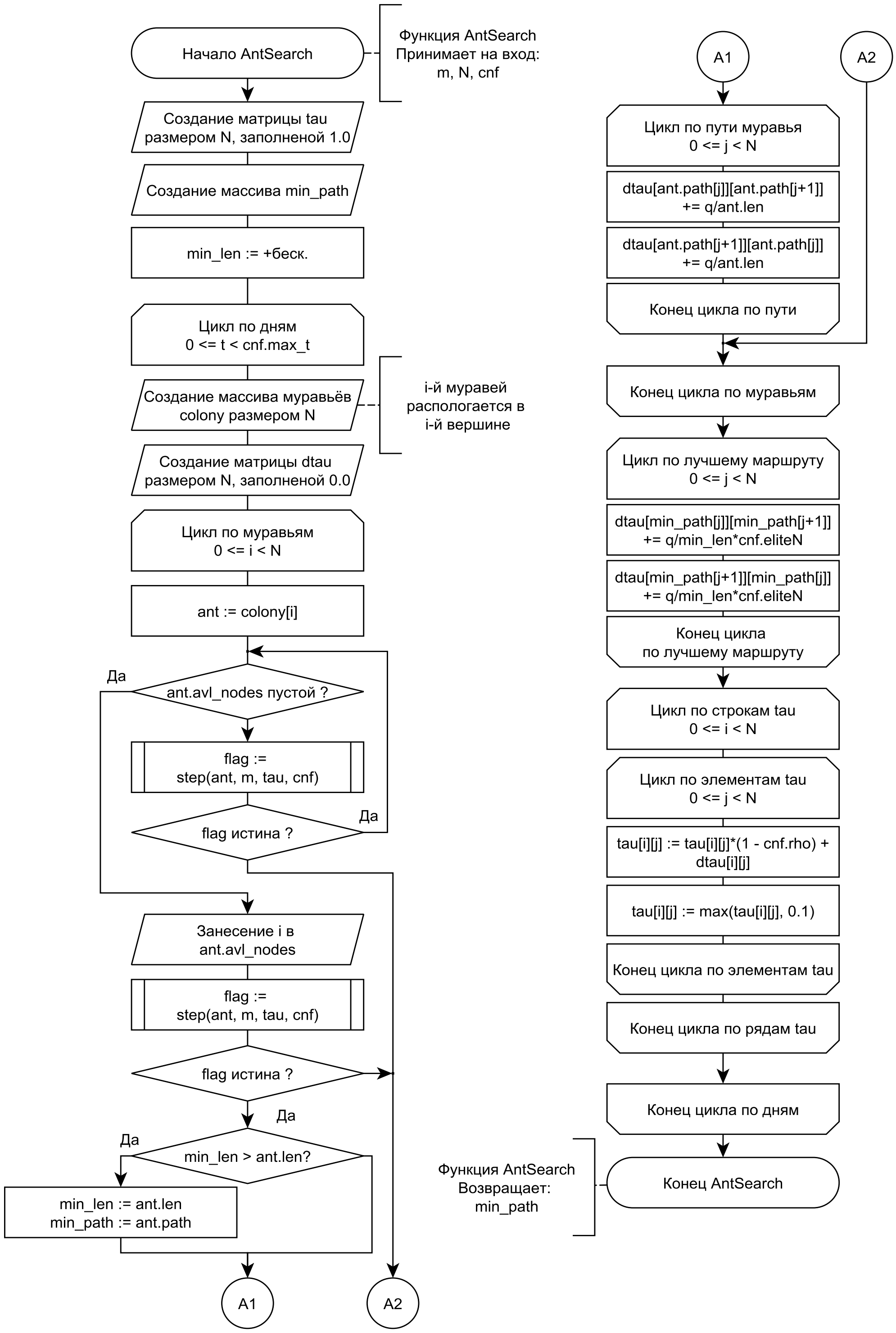

Рис. 2.2 — Поиск муравьиным алгоритмом (главная функция)

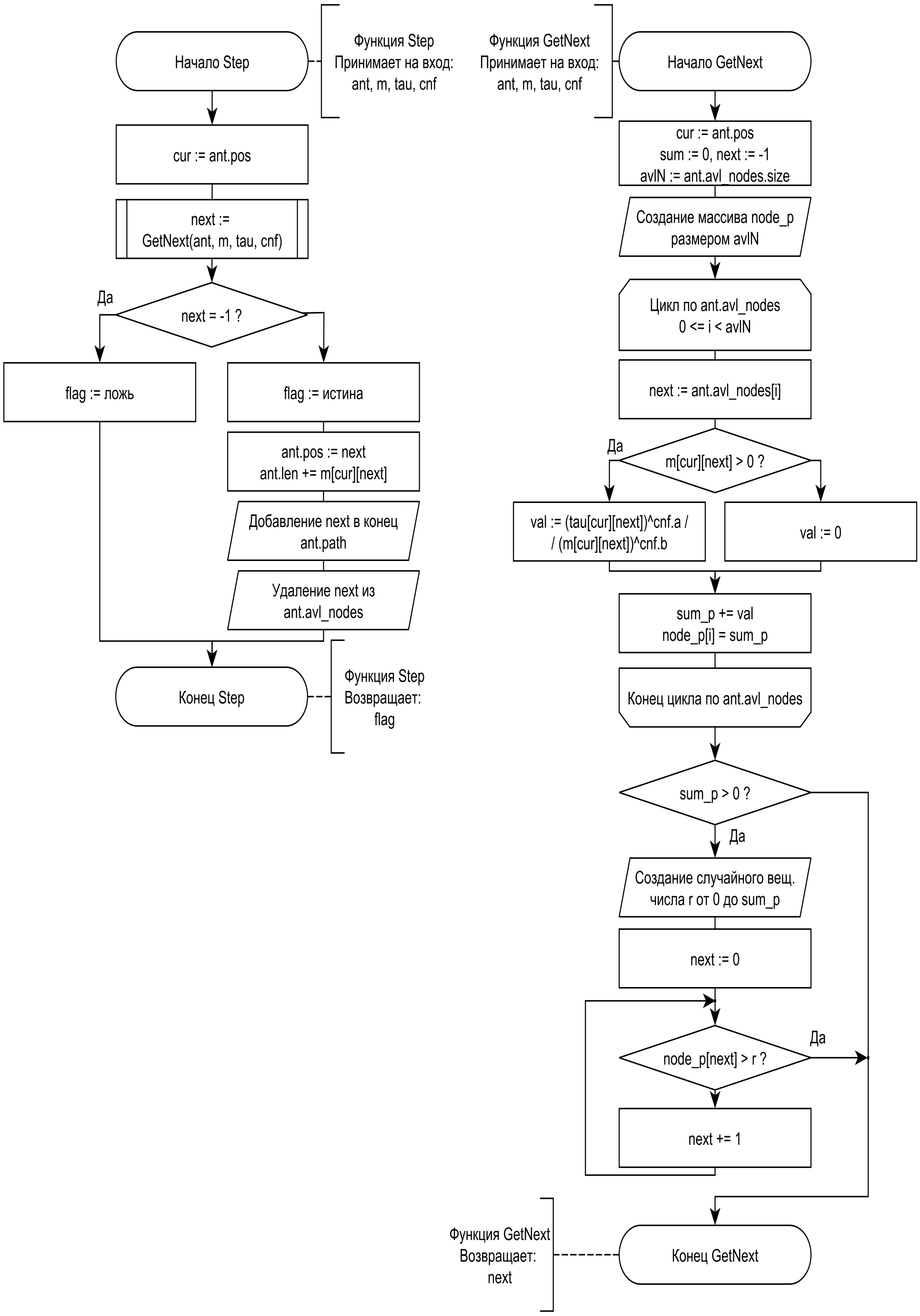

Рис. 2.3 — Поиск муравьиным алгоритмом (дополнительные функции)

3. Технологическая часть

3.1. Выбор языка программирования

В качестве языка программирования был выбран C++[2], так как имеется опыт работы с ним, и с библиотеками, позволяющими провести исследование и тестирование программы. Разработка проводилась в среде Visual Studio 2019[3].

3.2. Листинги кода

Реализация алгоритмов поиска представлена на листингах 3.1-3.3.

Листинг 3.1 — Поиск полным перебором

```
path_t explore_brunch(const len_matrix& m, const path_t&
    available_nodes, size_t cur_node, len_t& len)
{
  path_t res_path;
  len = -1;

  if (!available_nodes.size())
  {
    if (m[cur_node][0] < 0)
      return res_path;

    len = m[cur_node][0];
    res_path.push_back(0);
  }
  else
  {
    for (size_t i = 0; i < available_nodes.size(); i++)
    {
      size_t next_node = available_nodes[i];
      if (m[cur_node][next_node] < 0)
        continue;
```

```

      len_t temp_len = -1;
      path_t temp_nodes = available_nodes;
      temp_nodes.erase(temp_nodes.begin() + i);

      path_t temp_path = explore_brunch(m, temp_nodes, next_node, temp_len);
      if (temp_len < 0)
        continue;
      temp_len += m[cur_node][next_node];

      if (len < 0 || len > temp_len)
      {
        res_path = temp_path;
        len = temp_len;
      }
    }
  }

  res_path.push_back(cur_node);
  return res_path;
}

path_t brute_force(const len_matrix& m, len_t& len)
{
  path_t full_p(all_nodes(m));
  full_p.erase(full_p.begin()); // delete 0 node
  len = -1;
  path_t ans(explore_brunch(m, full_p, 0, len));
  if (ans.size() == m.size() + 1)
    return ans;
  else
  {
    len = 0;
```

```
    return path_t();
  }
}
```

Листинг 3.2 — Поиск муравьиным алгоритмом (главная функция)

```
path_t ant_search(const len_matrix& m, ant_config& cnf)
{
  double init_tau = 1;
  double min_tau = init_tau/10;
  vector<vector<double>> tau = create_matrix(m.size(),
   init_tau);

  path_t min_path;
  len_t min_len = -1;
  size_t elite_n = 4;

  for (size_t t = 0; t < cnf.max_t; t++)
  {
    ant_arr colony = init_colony(m);
    vector<vector<double>> d_tau = create_matrix(m.size(), 0)
    ;

    for (size_t i = 0; i < colony.size(); i++)
    {
      ant_t& ant = colony[i];
      size_t init_pos = ant.pos;

      while (ant.avl_nodes.size())
        if (!next_step(ant, m, tau, cnf))
          break;

      if (ant.avl_nodes.size())
      {
```

```
            ant.temp_len = -1;
        }
        else
        {
            ant.avl_nodes.push_back(init_pos);
            if (!next_step(ant, m, tau, cnf))
                ant.temp_len = -1;
        }
    }

    for (ant_t ant : colony)
    {
        if (ant.temp_len < 0)
            continue;

        if (min_len > ant.temp_len || min_len < 0)
        {
            min_len = ant.temp_len;
            min_path = ant.path;
        }

        double inc = ((double)cnf.q) / ant.temp_len;
        for (size_t i = 1; i < ant.path.size(); i++)
        {
            d_tau[ant.path[i]][ant.path[i-1]] += inc;
            d_tau[ant.path[i-1]][ant.path[i]] += inc;
        }
    }

    double inc = ((double)cnf.q) / min_len * elite_n;
    for (size_t i = 1; i < min_path.size(); i++)
    {
        d_tau[min_path[i]][min_path[i - 1]] += inc;
        d_tau[min_path[i - 1]][min_path[i]] += inc;
```

```
    }

    for (size_t i = 0; i < tau.size(); i++)
      for (size_t j = 0; j < tau.size(); j++)
        tau[i][j] = max(tau[i][j]*(1 - cnf.ro) + d_tau[i][j],
    min_tau);
  }

  return min_path;
}
```

Листинг 3.3 — Поиск муравьиным алгоритмом (дополнительные функции)

```
ant_arr init_colony(const len_matrix& m)
{
  size_t len = m.size();
  path_t pos = all_nodes(m);
  random_shuffle(pos.begin(), pos.end());

  ant_arr arr(len);
  for (size_t i = 0; i < len; i++)
  {
    arr[i].pos = pos[i];
    arr[i].temp_len = 0;
    arr[i].path.push_back(pos[i]);

    arr[i].avl_nodes = all_nodes(m);
    arr[i].avl_nodes.erase(arr[i].avl_nodes.begin() + pos[i])
   ;
  }

  return arr;
}
```

```
int get_next_node(const ant_t& ant, const len_matrix& m,
    const vector<vector<double>>& tau, const ant_config& cnf)
{
  size_t cur = ant.pos;
  vector<double> node_p(ant.avl_nodes.size(), 0);
  double sum_p = 0;

  for (size_t j = 0; j < ant.avl_nodes.size(); j++)
  {
    size_t next = ant.avl_nodes[j];
    if (m[cur][next] < 0)
      continue;

    double val = pow(tau[cur][next], cnf.a) / pow(m[cur][next
    ], cnf.b);
    sum_p += val;
    node_p[j] = sum_p;
  }
  if (sum_p < 1e-9)
    return -1;

  double rand_f = ((double)rand() / RAND_MAX) * sum_p * (1 -
    1e-8);

  for (size_t next = 0; next < node_p.size(); next++)
    if (node_p[next] > rand_f)
      return ant.avl_nodes[next];
  return ant.avl_nodes[node_p.size() - 1];
}
int next_step(ant_t& ant, const len_matrix& m, const vector<
    vector<double>>& tau, const ant_config& cnf)
{
  size_t cur = ant.pos;
```

```
  int next = get_next_node(ant, m, tau, cnf);
  if (next == -1) return 0;

  ant.pos = next;
  ant.temp_len += m[cur][next];
  ant.path.push_back(next);
  ant.avl_nodes.erase(find(ant.avl_nodes.begin(), ant.
   avl_nodes.end(), next));

  return 1;
}
```

3.3. Автоматическая параметризация муравьиного алгоритма

Для исследования работы муравьиного алгоритма на разных наборах функций была написана функция автоматической параметризации, приведённая в листинге 3.4.

Листинг 3.4 — Функции автоматической параметризации муравьиного алгоритма

```
void best_config(const len_matrix& m, len_t q, len_t
   perfect_len)
{
  len_t min_len = q * 1000;
  ant_config best_cnf;
  for (double a = 0; a <= 1; a+=0.1)
  {
    double b = 1 - a;
    for (double ro = 0; ro <= 1; ro += 0.1)
    {
      ant_config cnf = create_config(a, ro, 30, q);
```

```
            len_t local_min = q * 1000;
            for (int i=0; i<3; i++)
            {
                path_t p = ant_search(m, cnf);
                len_t len = path_len(m, p);
                local_min = min(len, local_min);
            }
            printf("%.1lf %.1lf %.1lf %zd: %.2lf %.2lf\n", a, b, ro, cnf.max_t, local_min, local_min - perfect_len);
            if (min_len > local_min)
            {
                min_len = local_min;
                best_cnf = cnf;
            }
        }
        printf("\n");
    }

    printf("%.1lf %.1lf %.1lf : %.2lf\n", best_cnf.a, best_cnf.b, best_cnf.ro, min_len);
}
```

3.4. Результаты тестирования

Для тестирования написанных функций был создан отдельный файл с ранее описанными классами тестов. Тестирование функций проводилось за счёт сравнения результов функций с ожидаемым результатом. Отдельно стоит отметить, что тестирование муравьиного алгоритма в общем случае затруднено непредсказуемостью ответа из-за случайной составляющей.

Состав тестов приведён в листинге 3.5.

Листинг 3.5 — Модульные тесты

```
#include "tests.h"
using namespace std;
bool _no_way()
{
  cout << __FUNCTION__;
  len_t len = 0;
  path_t path;
  len_matrix m = random_matrix(7, 1, 9, 0.99);

  len = 0;
  path = brute_force(m, len);
  if (len || path.size()) return false;

  ant_config cnf = create_config(0.5, 0.5, 20, calculate_q(m)
    );
  path = ant_search(m, cnf);
  if (path.size())    return false;
  return true;
}

bool _same_way()
{
  cout << __FUNCTION__;
  len_t len = 0;
  path_t path;
  len_matrix m = random_matrix(7, 1, 1);

  len = 0;
  path = brute_force(m, len);
  if (len != m.size()|| path.size() != m.size() + 1) return
    false;

```

```
  ant_config cnf = create_config(0.5, 0.5, 20, calculate_q(m));
  path = ant_search(m, cnf);
  if (path_len(m, path) != m.size() || path.size() != m.size() + 1)    return false;
  return true;
}

bool _size_two()
{
  cout << __FUNCTION__;
  len_t len = 0;
  path_t path;
  len_matrix m = random_matrix(2, 1, 9);

  len_t ans = m[1][0] + m[0][1];
  len = 0;
  path = brute_force(m, len);
  if (len != ans || path.size() != 3) return false;

  ant_config cnf = create_config(0.5, 0.5, 20, calculate_q(m));
  path = ant_search(m, cnf);
  if (path.size() != 3 || path_len(m, path) != ans)    return false;
  return true;
}

bool _rnd_matrix()
{
  cout << __FUNCTION__;
  len_t len = 0;
  path_t path;
  len_matrix m = random_matrix(10, 1, 9);
```

```
  len = 0;
  path = brute_force(m, len);
  if (path.size() != 11) return false;

  return true;
}


using test_f = bool(*)(void);
void run_tests()
{
  cout << "Running tests:" << endl;

  test_f f_arr[] = { _no_way, _same_way, _size_two,
    _rnd_matrix };

  for (size_t i = 0; i < 4; i++)
  {
    if (f_arr[i]())
      cout << " - PASSED\n";
    else
      cout << " - FAILED\n";
  }

  cout << endl;
}
```

3.5. Оценка времени

Для замера процессорного времени исполнения функции используется функция QueryPerformanceCounter библиотеки windows.h[4]. Код функций замера времени приведёны в листинге 3.6.

Листинг 3.6 — Функции замера процессорного времени работы функции

```
double PCFreq = 0.0;
__int64 CounterStart = 0;

void start_counter()
{
  LARGE_INTEGER li;
  QueryPerformanceFrequency(&li);

  PCFreq = double(li.QuadPart) / 1000.0;

  QueryPerformanceCounter(&li);
  CounterStart = li.QuadPart;
}

double get_counter()
{
  LARGE_INTEGER li;
  QueryPerformanceCounter(&li);
  return double(li.QuadPart - CounterStart) / PCFreq;
}
```

Вывод

Результатом технологической части стал выбор используемых технических средств реализации и реализация алгоритмов, системы тестов и замера времени работы на языке C++.

4. Исследовательская часть

4.1. Описание экспериментов

Исследование параметризации проводилось на графе из 10 вершин для трёх случаев:

1. значения длин - целые числа $\in [1, 10]$, все вершины соединены рёбрами;

2. значения длин - целые числа $\in [1, 10]$, примерно 25% вершины не соединены рёбрами;

3. значения длин - целые числа $\in [200, 400]$, все вершины соединены рёбрами.

Для повышения точности, каждый замер производится три раза, за результат берётся наикратчайший путь.

Также для муравьиного алгоритма проводится измерение времени процессорной работы для следующих размеров графа: $10, 20, 40, 80, 160$. Измерения проводятся с целью экспериментального установления трудоёмкости алгоритма в нотации O-большое. Для повышения точности, каждый замер производится пять раз, за результат берётся среднее арифметическое.

4.2. Эксперимент параметризации №1

Иследование проводилось по матрице расстояний 4.1.

$$\begin{bmatrix} 0 & 32 & 33 & 87 & 54 & 12 & 45 & 8 & 95 & 24 \\ 32 & 0 & 11 & 32 & 55 & 24 & 34 & 81 & 25 & 31 \\ 33 & 11 & 0 & 23 & 86 & 51 & 30 & 72 & 38 & 41 \\ 87 & 32 & 23 & 0 & 79 & 85 & 91 & 93 & 86 & 34 \\ 54 & 55 & 86 & 79 & 0 & 84 & 82 & 1 & 56 & 17 \\ 12 & 24 & 51 & 85 & 84 & 0 & 72 & 50 & 88 & 48 \\ 45 & 34 & 30 & 91 & 82 & 72 & 0 & 69 & 21 & 35 \\ 8 & 81 & 72 & 93 & 1 & 50 & 69 & 0 & 47 & 14 \\ 95 & 25 & 38 & 86 & 56 & 88 & 21 & 47 & 0 & 63 \\ 24 & 31 & 41 & 34 & 17 & 48 & 35 & 14 & 63 & 0 \end{bmatrix} \tag{4.1}$$

Метод полного перебора определил длину эталонного пути равную 195. По результатам параметризации можно составить таблицу 4.1

Таблица 4.1 — Результат параметризации №1

α	ρ	Количество итераций	Длина маршрута	Δ с эталоном
0,0	0,0	25	211	16
0,0	0,2	25	225	30
0,0	0,4	25	195	0
0,0	0,6	25	195	0
0,0	0,8	25	195	0
0,0	1,0	25	195	0
0,1	0,0	25	195	0
0,1	0,2	25	195	0
0,1	0,4	25	212	17
0,1	0,6	25	195	0
0,1	0,8	25	238	43
0,1	1,0	25	212	17
0,2	0,0	25	211	16
0,2	0,2	25	195	0
0,2	0,4	25	195	0
0,2	0,6	25	211	16
0,2	0,8	25	195	0
0,2	1,0	25	211	16
0,3	0,0	25	195	0
0,3	0,2	25	195	0
0,3	0,4	25	195	0
0,3	0,6	25	195	0
0,3	0,8	25	195	0
0,3	1,0	25	195	0
0,4	0,0	25	195	0
0,4	0,2	25	195	0
0,4	0,4	25	212	17
0,4	0,6	25	195	0
0,4	0,8	25	195	0
0,4	1,0	25	195	0
0,5	0,0	25	195	0
0,5	0,2	25	195	0
0,5	0,4	25	195	0
0,5	0,6	25	195	0
0,5	0,8	25	195	0
0,5	1,0	25	195	0
0,6	0,0	25	195	0

Продолжение на следующей странице

Таблица 4.1 – продолжение

α	ρ	Количество итераций	Длина маршрута	Δ с эталоном
0,6	0,2	25	195	0
0,6	0,4	25	195	0
0,6	0,6	25	195	0
0,6	0,8	25	195	0
0,6	1,0	25	195	0
0,7	0,0	25	231	36
0,7	0,2	25	195	0
0,7	0,4	25	195	0
0,7	0,6	25	195	0
0,7	0,8	25	195	0
0,7	1,0	25	195	0
0,8	0,0	25	195	0
0,8	0,2	25	211	16
0,8	0,4	25	195	0
0,8	0,6	25	195	0
0,8	0,8	25	195	0
0,8	1,0	25	195	0
0,9	0,0	25	225	30
0,9	0,2	25	211	16
0,9	0,4	25	195	0
0,9	0,6	25	212	17
0,9	0,8	25	195	0
0,9	1,0	25	211	16
1,0	0,0	25	212	17
1,0	0,2	25	211	16
1,0	0,4	25	231	36
1,0	0,6	25	211	16
1,0	0,8	25	195	0
1,0	1,0	25	195	0

4.3. Эксперимент параметризации №2

Иследование проводилось по матрице расстояний 4.2.

$$\begin{bmatrix} 0 & 18 & 39 & -1 & 85 & 48 & 90 & 35 & 99 & 60 \\ 18 & 0 & 8 & 60 & 90 & 92 & 11 & 3 & 39 & 77 \\ 39 & 8 & 0 & 62 & -1 & 51 & 61 & 95 & 44 & -1 \\ -1 & 60 & 62 & 0 & -1 & 33 & 27 & 32 & 34 & 67 \\ 85 & 90 & -1 & -1 & 0 & -1 & 44 & 39 & 14 & 19 \\ 48 & 92 & 51 & 33 & -1 & 0 & 26 & 87 & 26 & 6 \\ 90 & 11 & 61 & 27 & 44 & 26 & 0 & 47 & 3 & 80 \\ 35 & 3 & 95 & 32 & 39 & 87 & 47 & 0 & 1 & -1 \\ 99 & 39 & 44 & 34 & 14 & 26 & 3 & 1 & 0 & 40 \\ 60 & 77 & -1 & 67 & 19 & 6 & 80 & -1 & 40 & 0 \end{bmatrix} \tag{4.2}$$

Метод полного перебора определил длину эталонного пути равную 193. По результатам параметризации можно составить таблицу 4.2

Таблица 4.2 — Результат параметризации №2

α	ρ	Количество итераций	Длина маршрута	Δ с эталоном
0,0	0,0	25	199	6
0,0	0,2	25	193	0
0,0	0,4	25	193	0
0,0	0,6	25	193	0
0,0	0,8	25	193	0
0,0	1,0	25	200	7
0,1	0,0	25	193	0
0,1	0,2	25	199	6
0,1	0,4	25	193	0
0,1	0,6	25	193	0
0,1	0,8	25	199	6
0,1	1,0	25	193	0
0,2	0,0	25	205	12
0,2	0,2	25	193	0
0,2	0,4	25	200	7
0,2	0,6	25	193	0
0,2	0,8	25	193	0
0,2	1,0	25	193	0
0,3	0,0	25	193	0
0,3	0,2	25	193	0

Продолжение на следующей странице

Таблица 4.2 – продолжение

α	ρ	Количество итераций	Длина маршрута	Δ с эталоном
0,3	0,4	25	193	0
0,3	0,6	25	193	0
0,3	0,8	25	193	0
0,3	1,0	25	193	0
0,4	0,0	25	193	0
0,4	0,2	25	193	0
0,4	0,4	25	193	0
0,4	0,6	25	193	0
0,4	0,8	25	193	0
0,4	1,0	25	193	0
0,5	0,0	25	193	0
0,5	0,2	25	193	0
0,5	0,4	25	193	0
0,5	0,6	25	193	0
0,5	0,8	25	193	0
0,5	1,0	25	193	0
0,6	0,0	25	193	0
0,6	0,2	25	193	0
0,6	0,4	25	193	0
0,6	0,6	25	193	0
0,6	0,8	25	193	0
0,6	1,0	25	193	0
0,7	0,0	25	193	0
0,7	0,2	25	193	0
0,7	0,4	25	193	0
0,7	0,6	25	193	0
0,7	0,8	25	193	0
0,7	1,0	25	193	0
0,8	0,0	25	193	0
0,8	0,2	25	193	0
0,8	0,4	25	193	0
0,8	0,6	25	193	0
0,8	0,8	25	193	0
0,8	1,0	25	193	0
0,9	0,0	25	199	6
0,9	0,2	25	193	0
0,9	0,4	25	193	0
0,9	0,6	25	193	0
0,9	0,8	25	193	0
0,9	1,0	25	193	0
1,0	0,0	25	193	0
1,0	0,2	25	193	0
1,0	0,4	25	193	0
1,0	0,6	25	200	7
1,0	0,8	25	216	23

Продолжение на следующей странице

Таблица 4.2 – продолжение

α	ρ	Количество итераций	Длина маршрута	Δ с эталоном
1,0	1,0	25	193	0

4.4. Эксперимент параметризации №3

Иследование проводилось по матрице расстояний 4.3.

$$\begin{bmatrix} 0 & 318 & 391 & 313 & 302 & 345 & 344 & 289 & 359 & 283 \\ 318 & 0 & 242 & 248 & 328 & 312 & 323 & 235 & 227 & 361 \\ 391 & 242 & 0 & 317 & 368 & 371 & 284 & 397 & 204 & 385 \\ 313 & 248 & 317 & 0 & 283 & 343 & 211 & 248 & 233 & 281 \\ 302 & 328 & 368 & 283 & 0 & 269 & 330 & 344 & 236 & 227 \\ 345 & 312 & 371 & 343 & 269 & 0 & 201 & 373 & 270 & 301 \\ 344 & 323 & 284 & 211 & 330 & 201 & 0 & 319 & 273 & 258 \\ 289 & 235 & 397 & 248 & 344 & 373 & 319 & 0 & 383 & 399 \\ 359 & 227 & 204 & 233 & 236 & 270 & 273 & 383 & 0 & 202 \\ 283 & 361 & 385 & 281 & 227 & 301 & 258 & 399 & 202 & 0 \end{bmatrix} \tag{4.3}$$

Метод полного перебора определил длину эталонного пути равную 2393. По результатам параметризации можно составить таблицу 4.3

Таблица 4.3 — Результат параметризации №3

α	ρ	Количество итераций	Длина маршрута	Δ с эталоном
0,0	0,0	35	2485	92
0,0	0,2	35	2514	121
0,0	0,4	35	2453	60
0,0	0,6	35	2438	45
0,0	0,8	35	2445	52
0,0	1,0	35	2462	69
0,1	0,0	35	2417	24
0,1	0,2	35	2453	60
0,1	0,4	35	2484	91
0,1	0,6	35	2473	80
0,1	0,8	35	2394	1
0,1	1,0	35	2478	85

Продолжение на следующей странице

Таблица 4.3 – продолжение

α	ρ	Количество итераций	Длина маршрута	Δ с эталоном
0,2	0,0	35	2436	43
0,2	0,2	35	2445	52
0,2	0,4	35	2474	81
0,2	0,6	35	2463	70
0,2	0,8	35	2474	81
0,2	1,0	35	2411	18
0,3	0,0	35	2451	58
0,3	0,2	35	2453	60
0,3	0,4	35	2397	4
0,3	0,6	35	2436	43
0,3	0,8	35	2411	18
0,3	1,0	35	2474	81
0,4	0,0	35	2497	104
0,4	0,2	35	2459	66
0,4	0,4	35	2436	43
0,4	0,6	35	2453	60
0,4	0,8	35	2436	43
0,4	1,0	35	2397	4
0,5	0,0	35	2462	69
0,5	0,2	35	2463	70
0,5	0,4	35	2393	0
0,5	0,6	35	2393	0
0,5	0,8	35	2479	86
0,5	1,0	35	2411	18
0,6	0,0	35	2393	0
0,6	0,2	35	2394	1
0,6	0,4	35	2393	0
0,6	0,6	35	2439	46
0,6	0,8	35	2436	43
0,6	1,0	35	2411	18
0,7	0,0	35	2438	45
0,7	0,2	35	2393	0
0,7	0,4	35	2394	1
0,7	0,6	35	2411	18
0,7	0,8	35	2394	1
0,7	1,0	35	2393	0
0,8	0,0	35	2394	1
0,8	0,2	35	2393	0
0,8	0,4	35	2463	70
0,8	0,6	35	2393	0
0,8	0,8	35	2393	0
0,8	1,0	35	2394	1
0,9	0,0	35	2393	0
0,9	0,2	35	2393	0
0,9	0,4	35	2393	0

Продолжение на следующей странице

Таблица 4.3 – продолжение

α	ρ	Количество итераций	Длина маршрута	Δ с эталоном
0,9	0,6	35	2393	0
0,9	0,8	35	2393	0
0,9	1,0	35	2393	0
1,0	0,0	35	2394	1
1,0	0,2	35	2393	0
1,0	0,4	35	2393	0
1,0	0,6	35	2393	0
1,0	0,8	35	2394	1
1,0	1,0	35	2393	0

4.5. Результат замеров времени

По результатам измерений процессорного времени можно составить таблицу 4.4

Таблица 4.4 — Результат измерений процессорного времени (в секундах)

Размер	10	20	40	80	160
Время	$1.7 \cdot 10^{-3}$	0.011	0.081	0.61	4.75

При увеличении размерности в 2 раза время работы увеличивается примерно в 7.5, что ближе всего подходит для нотации $O(n^3)$

4.6. Характеристики ПК

Эксперименты проводились на компьютере с характеристиками:

- ОС - Windows 10, 64 бит;
- Процессор - Intel Core i7 8550U (1800 МГц, 4 ядра, 8 логических процессоров);
- Объем ОЗУ: 8 ГБ.

Вывод

По результатам экспериментов можно заключить следующее.

- В графе с большим количеством рёбер муравьиный алгоритм проявляет себя немного хуже по сравнению с графом, где часть городов не связаны рёбрами. Это объясняется тем, что в подобных графах возможных комбинаций путей заметно меньше, что сужает диапазон возможных решений

- В третьем эксперименте параметризации заметно проявляется то, что алгоритм работает при большей степени стадности. В двух других экспериментах это выражено меньше, так как и при большой стпени жадности алгоритм успешно находит решения.

- В случаях с α, ρ равных 0 или 1 в среднем алгоритм работает хуже, чем при иных параметрах.

- В самом худшем из встреченных случаев, погрешность решения по сравнению с эталоном составила 10%. В среднем, погрешность составляет примерно 2%.

- Трудоёмкость алгоритма полного перебора - $O(n!)$. В соответствии с проведёнными испытаниями, трудоёмкость муравьиного алгоритма составляет $O(n^3)$.

Заключение

В ходе лабораторной работы достигнута поставленная цель: проведён сравнительный анализ метода полного перебора и эвристического метода на базе муравьиного алгоритма.

Была изучена и описана задача коммивояжёра. Также были реализован метод полного перебора и метод на базе муравьиного алгоритма для решения задачи коммивояжёра. Проведены замеры процессорного времени и оценена трудоёмкость муравьиного алгоритма. Также проведена параметризация муравьиного метода и на основании полученных результатов проведён сравнительный анализ.

Из проведённых экспериментов можно заключить следующее. Алгоритм полного перебора всегда выдаёт правильное решение, но область его применимости ограничена графами, количество вершин которого не превышает 15. Далее, время вычисления ответа на любом устройстве будет неудволетвроительно долгим практически для любой задачи.

Алгоритм муравьиного поиска не является абсолютно точным, однако при правильно подобраных параметрах он способен находить точное решение, или маршрут, длина которого будет отличаться от эталонной на незначительную величину. При этом сложность алгоритма составляет примерно $O(n^3)$, поэтому он способен работать с графами, размер которых существенно превышает ограничение озвученное для предыдущего алгоритма.

Список литературы

1. Алгоритмы. Построение и анализ : пер. с анг. / Кормен Т., Лейзерсон Ч., Ривест Р. [и др.]. - 3-е изд. - М. : Вильямс, 2018. - 1323 с. : ил.

2. Документация языка C++ 98 [Электронный ресурс]. Режим доступа: http://www.open-std.org/JTC1/SC22/WG21/, свободный (дата обращения: 14.11.2020)

3. Документация среды разработки Visual Studio 2019 [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2019, свободный (дата обращения: 14.11.2020)

4. QueryPerformanceCounter function [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter, свободный (дата обращения: 29.10.2020).