



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 (ИУ)

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

База данных для транспортной системы завода

Студент

подпись, дата

Иванов В.А.

фамилия, и.о.

Руководитель курсовой работы

подпись, дата

Исаев А.Л.

фамилия, и.о.

2021 г.



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)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ7
(Индекс)
И.В. Рудаков
(И.О.Фамилия)
« ____ » _____ 2021 г.

З А Д А Н И Е на выполнение курсового проекта

по дисциплине Базы данных

Студент группы ИУ7-62Б

Иванов Всеволод Алексеевич
(Фамилия, имя, отчество)

Тема курсового проекта База данных для транспортной системы завода

Направленность КП (учебный, исследовательский, практический, производственный, др.)
учебный

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание Спроектировать и реализовать базу данных для транспортной системы завода. С помощью Web-приложения реализовать интерфейс для работы с информацией о заказах, записях о поездках, сотрудниках, дежурствах путём взаимодействия с базой данных. Также реализовать функционал для разных категорий пользователей

Оформление курсового проекта:

Расчетно-пояснительная записка на 20-30 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку задачи, введение, аналитическую, конструкторскую, технологическую части, заключение, список литературы.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.): на защиту проекта должна быть предоставлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграммы, интерфейс.

Дата выдачи задания « ____ » _____ 2021 г.

Руководитель курсового проекта

А.Л. Исаев
(Подпись, дата) (И.О.Фамилия)

Студент

В.А. Иванов
(Подпись, дата) (И.О.Фамилия)

Реферат

Курсовой проект заключается в реализации web-приложения для транспортной системы завода. В результате выполнения проекта было получено программное обеспечение предоставляющее возможность выполнять действия, необходимые для планирования и исполнения заказов доставки.

Приложение реализовано с использованием фреймворка Django на языке программирования Python 3. В качестве СУБД был использован PostgreSQL

Ключевые слова: доставка, транспортная система, завод, web-приложение, PostgreSQL, Django.

РПЗ содержит 37 страниц, 0 ссылок на используемые источники

Оглавление

Введение	6
1 Аналитическая часть	7
1.1 Постановка задачи	7
1.2 Формализация данных	8
1.3 Формализация ролей	8
1.4 Базы данных	9
1.4.1 Модели данных	9
1.4.2 СУБД	10
2 Конструкторская часть	12
2.1 Поиск полным перебором	12
2.2 Поиск муравьиным алгоритмом	12
2.3 Автоматическая параметризация муравьиного алгорит- ма	13
2.4 Требования к программному обеспечению	13
2.5 Заготовки тестов	14
3 Технологическая часть	15
3.1 Выбор языка программирования	15
3.2 Листинги кода	15
3.3 Автоматическая параметризация муравьиного алгорит- ма	21
3.4 Результаты тестирования	22
3.5 Оценка времени	25
3.6 Описание экспериментов	27
3.7 Эксперимент параметризации №1	27
3.8 Эксперимент параметризации №2	30
3.9 Эксперимент параметризации №3	32

3.10	Результат замеров времени	34
3.11	Характеристики ПК	34
	Заключение	36
	Список литературы	37

Введение

Одним из ключевых аспектов эффективного ведения любого крупного или среднего бизнеса в текущее время является хорошая информационная связанность между его сотрудниками. Быстрый и удобный доступ к данным позволяет оперативно координировать всех работников и минимизировать временные издержки по передаче актуальных задач. Поэтому, актуальной задачей в современном мире является разработка сервисов, реализующих данные возможности. Объектом разработки данного курсового проекта выбрана транспортная система завода, которая должна выполнять доставку заказов.

Целью данной курсовой работы является разработка базы данных для транспортной системы завода и web-приложения доступа к базе данных.

Выделены следующие задачи курсового проекта:

- формализовать задание, определить необходимый функционал;
- для хранения и структурирования данных спроектировать базу данных;
- проанализировать существующие СУБД и обосновать выбор одной из них;
- реализовать базу данных и интерфейс доступа к ней с использованием выбранной СУБД;
- реализовать web-приложение для взаимодействия с данными.

1. Аналитическая часть

1.1. Постановка задачи

Необходимо разработать web-приложение, которое будет предоставлять интерфейс для работы с информацией о сотрудниках, заказах, записях, дежурствах и о проездах машин через контрольно-пропускные пункты путём взаимодействия с базой данных.

Также должны быть реализованы различные категории сотрудников, для которых будет предоставляться определённый функционал по работе с данными транспортной системы. Для этого должна существовать возможность регистрации аккаунтов и дальнейшего взаимодействия с ними.

1.2. Формализация данных

В соответствии с предметной областью, соответствующей описанному набором требований, база данных должна хранить сущности, описанные ER-диаграммой на рисунке 1.1

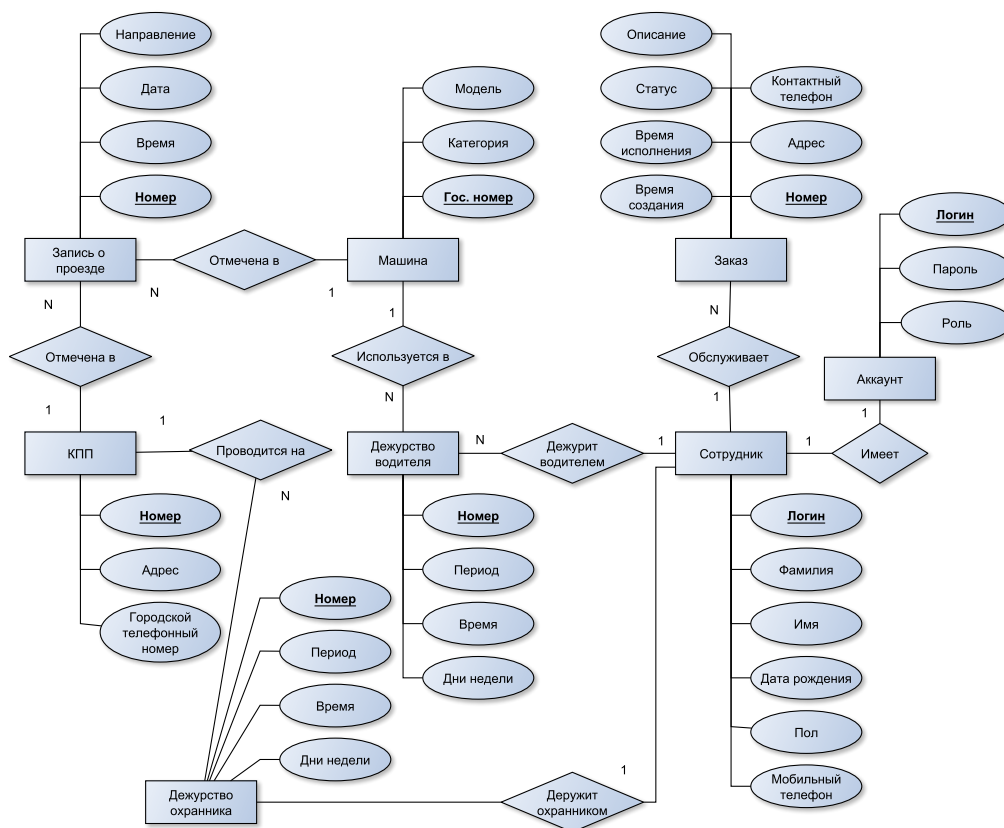


Рис. 1.1 — ER-диаграмма сущностей

1.3. Формализация ролей

Для функционирования транспортной системы были выделено несколько должностей сотрудников, название и функционал которых приведены в таблице 1.1.

Таблица 1.1 — роли сотрудников и их функционал

Должность	Функционал
Администратор	Назначение дежурств и заказов, рассмотрение заявок на регистрацию, создание новых сущностей: машин, КПП, заказов
Водитель	Выбор заказа, выполнение заказа, просмотр личной информации
Охранник	Создание записей о проезде, просмотр личной информации
Неподтверждённый	Просмотр личной информации
Неавторизованный	Регистрация, авторизация

Для упрощения регистрации новых сотрудников в системе принято решение предоставить им возможность заполнения заявок, которые в дальнейшем могут быть подтверждены администратором.

1.4. Базы данных

База данных — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных [1]

1.4.1. Модели данных

Модель данных - это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных. [1]

Существует три основные модели данных.

- Иерархическая. База данных представляется в виде древо-

видной структуры, состоящей из объектов различных уровней. Объекты находятся в отношении предка к потомку, причём объект может иметь только одного предка и любое количество потомков.

- Сетевая. Отличием от иерархической модели данных является отсутствие ограничения на количество предков. База данных в такой модели состоит из набора экземпляров записи и экземпляров связей между записями.
- Реляционная. Основной идеей является то, что все наборы данных представляются в виде множеств. База данных состоит из множества двумерных таблиц. Таблицы состоят из записей и полей, каждая запись имеет собственные значения для каждого поля. Иерархия элементов отсутствует.

Реляционная модель в настоящее время является наиболее гибкой и удобной в использовании, а также обладает наибольшим выбором в области систем управления базой данных, поэтому принято решение о её использовании в данной работе.

1.4.2. СУБД

Система управления базами данных (сокращённо СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных [1].

В процессе разработки было принято решение использовать СУБД для взаимодействия с базой данных.

Вывод

Результатом аналитического раздела стала постановка задачи, формализованны данные программы и роли пользователей, описаны способы хранения данных и управления ими.

2. Конструкторская часть

Рассмотрим описанные алгоритмы поиска маршрута. Пусть поиск производится по квадратной матрице расстояний 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 найденных 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.

2.2. Поиск муравьиным алгоритмом

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

Каждый муравей содержит информацию 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 най-

денными 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 $d\tau$ приращения феромонов и её заполнение в соответствии с пройденными путями. Также симулируется прохождение элитных муравьёв по текущему лучшему пути. После этого производится испарение феромона и занесение значений из $d\tau$ в τ . Контролируется итоговое значение ячеек τ – оно не должно опускаться ниже 0.1 от начальной величины.

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 конструкторской части стало схематическое описание алгоритмов поиска и параметризации, сформулированные тесты и требования к программному обеспечению.

3. Технологическая часть

3.1. Выбор языка программирования

В качестве языка программирования был выбран C++[?], 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

```
1 path_t explore_brunch(const len_matrix& m, const path_t&
   available_nodes, size_t cur_node, len_t& len)
2 {
3     path_t res_path;
4     len = -1;
5
6     if (!available_nodes.size())
7     {
8         if (m[cur_node][0] < 0)
9             return res_path;
10
11         len = m[cur_node][0];
12         res_path.push_back(0);
13     }
14     else
15     {
16         for (size_t i = 0; i < available_nodes.size(); i++)
17         {
18             size_t next_node = available_nodes[i];
19             if (m[cur_node][next_node] < 0)
20                 continue;
```

```

21
22     len_t temp_len = -1;
23     path_t temp_nodes = available_nodes;
24     temp_nodes.erase(temp_nodes.begin() + i);
25
26     path_t temp_path = explore_brunch(m, temp_nodes,
next_node, temp_len);
27     if (temp_len < 0)
28         continue;
29     temp_len += m[cur_node][next_node];
30
31     if (len < 0 || len > temp_len)
32     {
33         res_path = temp_path;
34         len = temp_len;
35     }
36 }
37 }
38
39 res_path.push_back(cur_node);
40 return res_path;
41 }
42
43 path_t brute_force(const len_matrix& m, len_t& len)
44 {
45     path_t full_p(all_nodes(m));
46     full_p.erase(full_p.begin()); // delete 0 node
47     len = -1;
48     path_t ans(explore_brunch(m, full_p, 0, len));
49     if (ans.size() == m.size() + 1)
50         return ans;
51     else
52     {
53         len = 0;

```



```

54     return path_t();
55 }
56 }

```

Листинг 3.2 — Поиск муравьиным алгоритмом (главная функция)

```

1 path_t ant_search(const len_matrix& m, ant_config& cnf)
2 {
3     double init_tau = 1;
4     double min_tau = init_tau / 10;
5     vector<vector<double>> tau = create_matrix(m.size(),
6         init_tau);
7
8     path_t min_path;
9     len_t min_len = -1;
10    size_t elite_n = 4;
11
12    for (size_t t = 0; t < cnf.max_t; t++)
13    {
14        ant_arr colony = init_colony(m);
15        vector<vector<double>> d_tau = create_matrix(m.size(), 0);
16
17        for (size_t i = 0; i < colony.size(); i++)
18        {
19            ant_t& ant = colony[i];
20            size_t init_pos = ant.pos;
21
22            while (ant.avl_nodes.size())
23                if (!next_step(ant, m, tau, cnf))
24                    break;
25
26            if (ant.avl_nodes.size())
27            {

```

```

27     ant.temp_len = -1;
28 }
29 else
30 {
31     ant.avl_nodes.push_back(init_pos);
32     if (!next_step(ant, m, tau, cnf))
33         ant.temp_len = -1;
34 }
35 }
36
37 for (ant_t ant : colony)
38 {
39     if (ant.temp_len < 0)
40         continue;
41
42     if (min_len > ant.temp_len || min_len < 0)
43     {
44         min_len = ant.temp_len;
45         min_path = ant.path;
46     }
47
48     double inc = ((double)cnf.q) / ant.temp_len;
49     for (size_t i = 1; i < ant.path.size(); i++)
50     {
51         d_tau[ant.path[i]][ant.path[i-1]] += inc;
52         d_tau[ant.path[i-1]][ant.path[i]] += inc;
53     }
54 }
55
56 double inc = ((double)cnf.q) / min_len * elite_n;
57 for (size_t i = 1; i < min_path.size(); i++)
58 {
59     d_tau[min_path[i]][min_path[i-1]] += inc;
60     d_tau[min_path[i-1]][min_path[i]] += inc;

```

```

61     }
62
63     for (size_t i = 0; i < tau.size(); i++)
64         for (size_t j = 0; j < tau.size(); j++)
65             tau[i][j] = max(tau[i][j]*(1 - cnf.ro) + d_tau[i][j],
66                             min_tau);
67
68     return min_path;
69 }

```

Листинг 3.3 — Поиск муравьиным алгоритмом (дополнительные функции)

```

1 ant_arr init_colony(const len_matrix& m)
2 {
3     size_t len = m.size();
4     path_t pos = all_nodes(m);
5     random_shuffle(pos.begin(), pos.end());
6
7     ant_arr arr(len);
8     for (size_t i = 0; i < len; i++)
9     {
10         arr[i].pos = pos[i];
11         arr[i].temp_len = 0;
12         arr[i].path.push_back(pos[i]);
13
14         arr[i].avl_nodes = all_nodes(m);
15         arr[i].avl_nodes.erase(arr[i].avl_nodes.begin() + pos[i])
16         ;
17     }
18     return arr;
19 }

```

```

20
21 int get_next_node(const ant_t& ant, const len_matrix& m,
    const vector<vector<double>>& tau, const ant_config& cnf)
22 {
23     size_t cur = ant.pos;
24     vector<double> node_p(ant.avl_nodes.size(), 0);
25     double sum_p = 0;
26
27     for (size_t j = 0; j < ant.avl_nodes.size(); j++)
28     {
29         size_t next = ant.avl_nodes[j];
30         if (m[cur][next] < 0)
31             continue;
32
33         double val = pow(tau[cur][next], cnf.a) / pow(m[cur][next], cnf.b);
34         sum_p += val;
35         node_p[j] = sum_p;
36     }
37     if (sum_p < 1e-9)
38         return -1;
39
40     double rand_f = ((double)rand() / RAND_MAX) * sum_p * (1 -
        1e-8);
41
42     for (size_t next = 0; next < node_p.size(); next++)
43         if (node_p[next] > rand_f)
44             return ant.avl_nodes[next];
45     return ant.avl_nodes[node_p.size() - 1];
46 }
47 int next_step(ant_t& ant, const len_matrix& m, const vector<
    vector<double>>& tau, const ant_config& cnf)
48 {
49     size_t cur = ant.pos;

```

```

50  int next = get_next_node(ant, m, tau, cnf);
51  if (next == -1) return 0;
52
53  ant.pos = next;
54  ant.temp_len += m[cur][next];
55  ant.path.push_back(next);
56  ant.avl_nodes.erase(find(ant.avl_nodes.begin(), ant.
    avl_nodes.end(), next));
57
58  return 1;
59 }

```

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

```

1  void best_config(const len_matrix& m, len_t q, len_t
    perfect_len)
2  {
3      len_t min_len = q * 1000;
4      ant_config best_cnf;
5      for (double a = 0; a <= 1; a+=0.1)
6      {
7          double b = 1 - a;
8          for (double ro = 0; ro <= 1; ro += 0.1)
9          {
10             ant_config cnf = create_config(a, ro, 30, q);

```

```

11     len_t local_min = q * 1000;
12     for (int i=0; i<3; i++)
13     {
14         path_t p = ant_search(m, cnf);
15         len_t len = path_len(m, p);
16         local_min = min(len, local_min);
17     }
18     printf("%.1lf %.1lf %.1lf %zd: %.2lf %.2lf\n", a, b, ro
, cnf.max_t, local_min, local_min - perfect_len);
19     if (min_len > local_min)
20     {
21         min_len = local_min;
22         best_cnf = cnf;
23     }
24 }
25 printf("\n");
26 }
27
28 printf("%.1lf %.1lf %.1lf : %.2lf\n", best_cnf.a, best_cnf.
b, best_cnf.ro, min_len);
29 }

```

3.4. Результаты тестирования

Для тестирования написанных функций был создан отдельный файл с ранее описанными классами тестов. Тестирование функций проводилось за счёт сравнения результатов 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

```
1 #include "tests.h"
2 using namespace std;
3 bool _no_way()
4 {
5     cout << __FUNCTION__;
6     len_t len = 0;
7     path_t path;
8     len_matrix m = random_matrix(7, 1, 9, 0.99);
9
10    len = 0;
11    path = brute_force(m, len);
12    if (len || path.size()) return false;
13
14    ant_config cnf = create_config(0.5, 0.5, 20, calculate_q(m)
15    );
16    path = ant_search(m, cnf);
17    if (path.size()) return false;
18    return true;
19 }
20 bool _same_way()
21 {
22     cout << __FUNCTION__;
23     len_t len = 0;
24     path_t path;
25     len_matrix m = random_matrix(7, 1, 1);
26
27     len = 0;
28     path = brute_force(m, len);
29     if (len != m.size() || path.size() != m.size() + 1) return
30         false;
```

```

31 | ant_config cnf = create_config(0.5, 0.5, 20, calculate_q(m)
    | );
32 | path = ant_search(m, cnf);
33 | if (path_len(m, path) != m.size() || path.size() != m.size
    | () + 1) return false;
34 | return true;
35 | }
36 |
37 | bool _size_two()
38 | {
39 |     cout << __FUNCTION__;
40 |     len_t len = 0;
41 |     path_t path;
42 |     len_matrix m = random_matrix(2, 1, 9);
43 |
44 |     len_t ans = m[1][0] + m[0][1];
45 |     len = 0;
46 |     path = brute_force(m, len);
47 |     if (len != ans || path.size() != 3) return false;
48 |
49 |     ant_config cnf = create_config(0.5, 0.5, 20, calculate_q(m)
    | );
50 |     path = ant_search(m, cnf);
51 |     if (path.size() != 3 || path_len(m, path) != ans) return
    | false;
52 |     return true;
53 | }
54 |
55 | bool _rnd_matrix()
56 | {
57 |     cout << __FUNCTION__;
58 |     len_t len = 0;
59 |     path_t path;
60 |     len_matrix m = random_matrix(10, 1, 9);

```



```

61
62     len = 0;
63     path = brute_force(m, len);
64     if (path.size() != 11) return false;
65
66     return true;
67 }
68
69
70 using test_f = bool (*)(void);
71 void run_tests()
72 {
73     cout << "Running tests:" << endl;
74
75     test_f f_arr[] = { _no_way, _same_way, _size_two,
76                         _rnd_matrix };
77
78     for (size_t i = 0; i < 4; i++)
79     {
80         if (f_arr[i]())
81             cout << " — PASSED\n";
82         else
83             cout << " — FAILED\n";
84     }
85     cout << endl;
86 }

```

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

```
1 double PCFreq = 0.0;
2 __int64 CounterStart = 0;
3
4 void start_counter()
5 {
6     LARGE_INTEGER li;
7     QueryPerformanceFrequency(&li);
8
9     PCFreq = double(li.QuadPart) / 1000.0;
10
11     QueryPerformanceCounter(&li);
12     CounterStart = li.QuadPart;
13 }
14
15 double get_counter()
16 {
17     LARGE_INTEGER li;
18     QueryPerformanceCounter(&li);
19     return double(li.QuadPart - CounterStart) / PCFreq;
20 }
```

Вывод

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

3.6. 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.

3.7. Эксперимент параметризации №1

Исследование проводилось по матрице расстояний 3.1.

$$\begin{bmatrix} 0 & 32 & 33 & 87 & 54 & 12 & 45 & 8 & 95 & 24 \\ 32 & 0 & 11 & 32 & 55 & 24 & 34 & 81 & 25 & 31 \\ 33 & 11 & 0 & 23 & 86 & 51 & 30 & 72 & 38 & 41 \\ 87 & 32 & 23 & 0 & 79 & 85 & 91 & 93 & 86 & 34 \\ 54 & 55 & 86 & 79 & 0 & 84 & 82 & 1 & 56 & 17 \\ 12 & 24 & 51 & 85 & 84 & 0 & 72 & 50 & 88 & 48 \\ 45 & 34 & 30 & 91 & 82 & 72 & 0 & 69 & 21 & 35 \\ 8 & 81 & 72 & 93 & 1 & 50 & 69 & 0 & 47 & 14 \\ 95 & 25 & 38 & 86 & 56 & 88 & 21 & 47 & 0 & 63 \\ 24 & 31 & 41 & 34 & 17 & 48 & 35 & 14 & 63 & 0 \end{bmatrix} \quad (3.1)$$

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

Таблица 3.1 — 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
Продолжение на следующей странице				

Таблица 3.1 – продолжение

α	ρ	Количество итераций	Длина маршрута	Δ с эталоном
0,6	0,0	25	195	0
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

3.8. Эксперимент параметризации №2

Исследование проводилось по матрице расстояний 3.2.

$$\begin{bmatrix}
 0 & 18 & 39 & -1 & 85 & 48 & 90 & 35 & 99 & 60 \\
 18 & 0 & 8 & 60 & 90 & 92 & 11 & 3 & 39 & 77 \\
 39 & 8 & 0 & 62 & -1 & 51 & 61 & 95 & 44 & -1 \\
 -1 & 60 & 62 & 0 & -1 & 33 & 27 & 32 & 34 & 67 \\
 85 & 90 & -1 & -1 & 0 & -1 & 44 & 39 & 14 & 19 \\
 48 & 92 & 51 & 33 & -1 & 0 & 26 & 87 & 26 & 6 \\
 90 & 11 & 61 & 27 & 44 & 26 & 0 & 47 & 3 & 80 \\
 35 & 3 & 95 & 32 & 39 & 87 & 47 & 0 & 1 & -1 \\
 99 & 39 & 44 & 34 & 14 & 26 & 3 & 1 & 0 & 40 \\
 60 & 77 & -1 & 67 & 19 & 6 & 80 & -1 & 40 & 0
 \end{bmatrix} \quad (3.2)$$

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

Таблица 3.2 — 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 3.2 – 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 3.2 – продолжение

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

3.9. Эксперимент параметризации №3

Исследование проводилось по матрице расстояний 3.3.

$$\begin{bmatrix}
 0 & 318 & 391 & 313 & 302 & 345 & 344 & 289 & 359 & 283 \\
 318 & 0 & 242 & 248 & 328 & 312 & 323 & 235 & 227 & 361 \\
 391 & 242 & 0 & 317 & 368 & 371 & 284 & 397 & 204 & 385 \\
 313 & 248 & 317 & 0 & 283 & 343 & 211 & 248 & 233 & 281 \\
 302 & 328 & 368 & 283 & 0 & 269 & 330 & 344 & 236 & 227 \\
 345 & 312 & 371 & 343 & 269 & 0 & 201 & 373 & 270 & 301 \\
 344 & 323 & 284 & 211 & 330 & 201 & 0 & 319 & 273 & 258 \\
 289 & 235 & 397 & 248 & 344 & 373 & 319 & 0 & 383 & 399 \\
 359 & 227 & 204 & 233 & 236 & 270 & 273 & 383 & 0 & 202 \\
 283 & 361 & 385 & 281 & 227 & 301 & 258 & 399 & 202 & 0
 \end{bmatrix} \quad (3.3)$$

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

Таблица 3.3 — 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 3.3 – 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 3.3 – 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

3.10. Результат замеров времени

По результатам измерений процессорного времени можно составить таблицу 4.4

Таблица 3.4 — 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 $O(n^3)$

3.11. 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 степени 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 неудовлетворительно долгим практически для любой задачи.

Алгоритм муравьиного поиска не является абсолютно точным, однако при правильно подобранных 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. ISO/IEC TR 10032:2003 Information technology — Reference model of data management
2. Дейт К. Дж. Введение в системы баз данных. — 8-е изд. — М.: «Вильямс», 2006.
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-queryperformancenumerator>, свободный (дата обращения: 29.10.2020).