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"

Студент Сукочева Алис
Группа ИУ7-53Б
Название предприятия МГТУ им. Н. Э. Баумана, каф. ИУ7
Тема Муравьиный алгоритм.

Студент:	_____	Сукочева А.
	подпись, дата	Фамилия, И.О.
Преподаватель:	_____	Волкова Л.Л.
	подпись, дата	Фамилия, И. О.
Преподаватель:	_____	Строганов Ю.В.
	подпись, дата	Фамилия, И. О.

Москва — 2020 г.

Оглавление

Введение

Муравьиный алгоритм – алгоритм оптимизации подражанием муравьиной колонии. один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах.

Муравьиные алгоритмы серьезно исследуются европейскими учеными с середины 1990-х годов. Муравьиный алгоритм предоставляет хорошие результаты оптимизации для многих сложных комбинаторных задач, таких как:

- задачи коммивояжера;
- раскраски графа;
- оптимизации маршрутов грузовиков;
- квадратичной задачи о назначениях;
- оптимизации сетевых графиков;
- задачи календарного планирования.

Целью данной работы является изучение и реализация двух алгоритмов:

1. полный перебор;
2. муравьиный алгоритм.

В рамках выполнения работы необходимо решить следующие задачи:

1. изучить два, описанных выше, алгоритма для решения задачи коммивояжера;

2. применить изученные основы для реализации двух алгоритмов;
3. получить практические навыки;
4. провести параметризацию муравьиного алгоритма;
5. провести сравнительный анализ скорости работы реализованных алгоритмов;
6. выбрать и обосновать выбор язык программирования, для решения поставленной задачи.

1 Аналитическая часть

1.1 Задача коммивояжёра

Коммивояжёр (фр. commis voyageur) – бродячий торговец.

Задача коммивояжёра – важная задача транспортной логистики, отрасли, занимающейся планированием транспортных перевозок. Коммивояжёру, чтобы распродать нужные и не очень нужные в хозяйстве товары, следует объехать n пунктов и в конце концов вернуться в исходный пункт. Требуется определить наиболее выгодный маршрут объезда. В качестве меры выгодности маршрута (точнее говоря, невыгодности) может служить суммарное время в пути, суммарная стоимость дороги, или, в простейшем случае, длина маршрута.

1.2 Решение полным перебором

Задача может быть решена перебором всех вариантов объезда и выбором оптимального. Но при таком подходе количество возможных маршрутов очень быстро возрастает с ростом n (оно равно n! — количеству способов упорядочения пунктов). К примеру, для 100 пунктов количество вариантов будет представляться 158-значным числом. Мощная ЭВМ, способная перебирать миллион вариантов в секунду, будет решать такую задачу на протяжении примерно 3*10144 лет. Увеличение производительности ЭВМ в 1000 раз даст хоть и меньшее в 1000 раз, но по-прежнему очень большое время перебора вариантов.

Хотя такой подход и гарантирует точное решение задачи, уже при небольшом числе городов решение за допустимое количество времени невозможно.

1.3 Решение с помощью муравьиного алгоритма

В то время как простой метод перебора всех вариантов чрезвычайно неэффективный при большом количестве городов, эффективными признаются решения, гарантирующие получение ответа за время, ограниченное полиномом от размерности задачи.

В основе алгоритма лежит поведение муравьиной колонии – маркировка более удачных путей большим количеством феромона. Рассмотрим биологическую модель поведения такой колонии.

В реальном мире муравьи (первоначально) ходят в случайном порядке и по нахождении продовольствия возвращаются в свою колонию, прокладывая феромонами тропы. Если другие муравьи находят такие тропы, они, вероятнее всего, пойдут по ним. Вместо того, чтобы отслеживать цепочку, они укрепляют её при возвращении, если в конечном итоге находят источник питания. Со временем феромонная тропа начинает испаряться, тем самым уменьшая свою привлекательную силу. Чем больше времени требуется для прохождения пути до цели и обратно, тем сильнее испарится феромонная тропа. На коротком пути, для сравнения, прохождение будет более быстрым, и, как следствие, плотность феромонов остаётся высокой. Если бы феромоны не испарялись, то путь, выбранный первым, был бы самым привлекательным. В этом случае, исследования пространственных решений были бы ограниченными. Таким образом, когда один муравей находит (например, короткий) путь от колонии до источника пищи, другие муравьи, скорее всего пойдут по этому пути, и положительные отзывы в конечном итоге приводят всех муравьёв к одному, кратчайшему, пути. Этапы работы муравьиной колонии представлены на рис. 1.1.

Процесс поиска кратчайшего пути от колонии до источника питания (рис 1.1):

1. первый муравей находит источник пищи (F) любым способом (a), а затем возвращается к гнезду (N), оставив за собой тропу из феромонов (b);

2. затем муравьи выбирают один из четырёх возможных путей, затем укрепляют его и делают привлекательным;

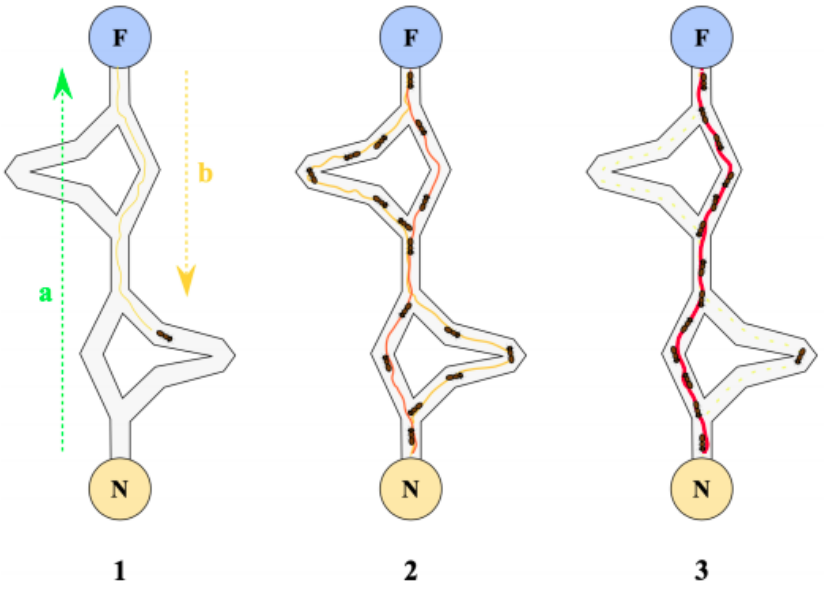

Рис. 1.1: Работа муравьиной колонии

3. муравьи выбирают кратчайший маршрут, так как феромоны с более длинных путей быстрее испаряются.

Вероятность перехода из вершины i в вершину j определяется по формуле 1.1.

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})} \tag{1.1}$$

где $\tau_{i,j}$— расстояние от города i до j;
$\eta_{i,j}$—количество феромонов на ребре ij;
α— параметр влияния длины пути;
β— параметр влияния феромона.
Уровень феромона обновляется в соответствии с формулой 1.2

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} + \Delta\tau_{i,j}, \tag{1.2}$$

где ρ - доля феромона, которая испарится;
$\tau_{i,j}$ - количество феромона на дуге ij;

$\Delta\tau_{i,j}$ - количество отложенного феромона, вычисляется по формуле 1.3.

$$\Delta\tau_{i,j} = \tau_{i,j}^0 + \tau_{i,j}^1 + ... + \tau_{i,j}^k \tag{1.3}$$

где k - количество муравьев в вершине графа с индексами i и j.
Описание поведения муравьев при выборе пути.

- Муравьи имеют собственную «память». Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещенных городов - список запретов. Обозначим через J_{ik} список городов, которые необходимо посетить муравью k, находящемуся в городе i.

- Муравьи обладают «зрением» - видимость есть эвристическое желание посетить город j , если муравей находится в городе i. Будем считать, что видимость обратно пропорциональна расстоянию между городами.

- Муравьи обладают «обонянием» - они могут улавливать след феромона, подтверждающий желание посетить город j из города i на основании опыта других муравьёв. Количество феромона на ребре (i, j) в момент времени t обозначим через $\tau_{i,j}(t)$.

- Пройдя ребро (i, j) , муравей откладывает на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть $T_k(t)$ есть маршрут, пройденный муравьем k к моменту времени t , $L_k(t)$ - длина этого маршрута, а Q - параметр, имеющий значение порядка длины оптимального пути. Тогда откладываемое количество феромона может быть задано формулой 1.4.

$$\Delta\tau_{i,j}^k = \begin{cases} Q/L_k, & \text{если k-ый мурваей прошел по ребру ij;} \\ 0, & \text{иначе} \end{cases} \tag{1.4}$$

где Q - количество феромона, переносимого муравьем.

1.4 Вывод

В данном разделе были рассмотрены основополагающие материалы, которые в дальнейшем потребуются при реализации алгоритмов для решения задачи коммивояжера.

2 | Конструкторская часть

2.1 Разработка алгоритмов

В данном разделе будут рассмотрены схемы.

На рис. 2.1 показана схема алгоритма полного перебора.

На рис. 2.2 показана схема муравьиного алгоритма.

2.2 Вывод

В данном разделе были рассмотрены схемы алгоритма полного перебора 2.1 и схема муравьиного алгоритма 2.2.

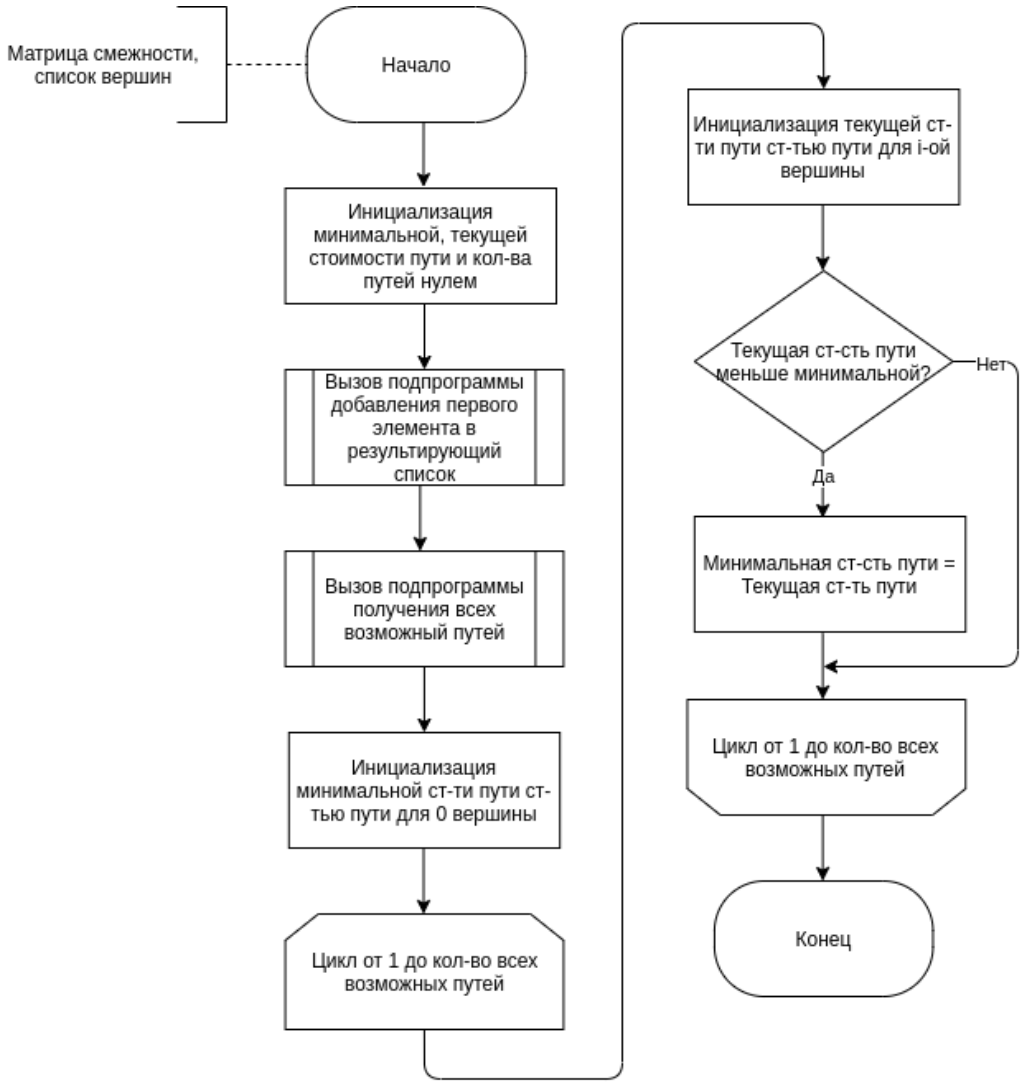

Рис. 2.1: Схема алгоритма полного перебора

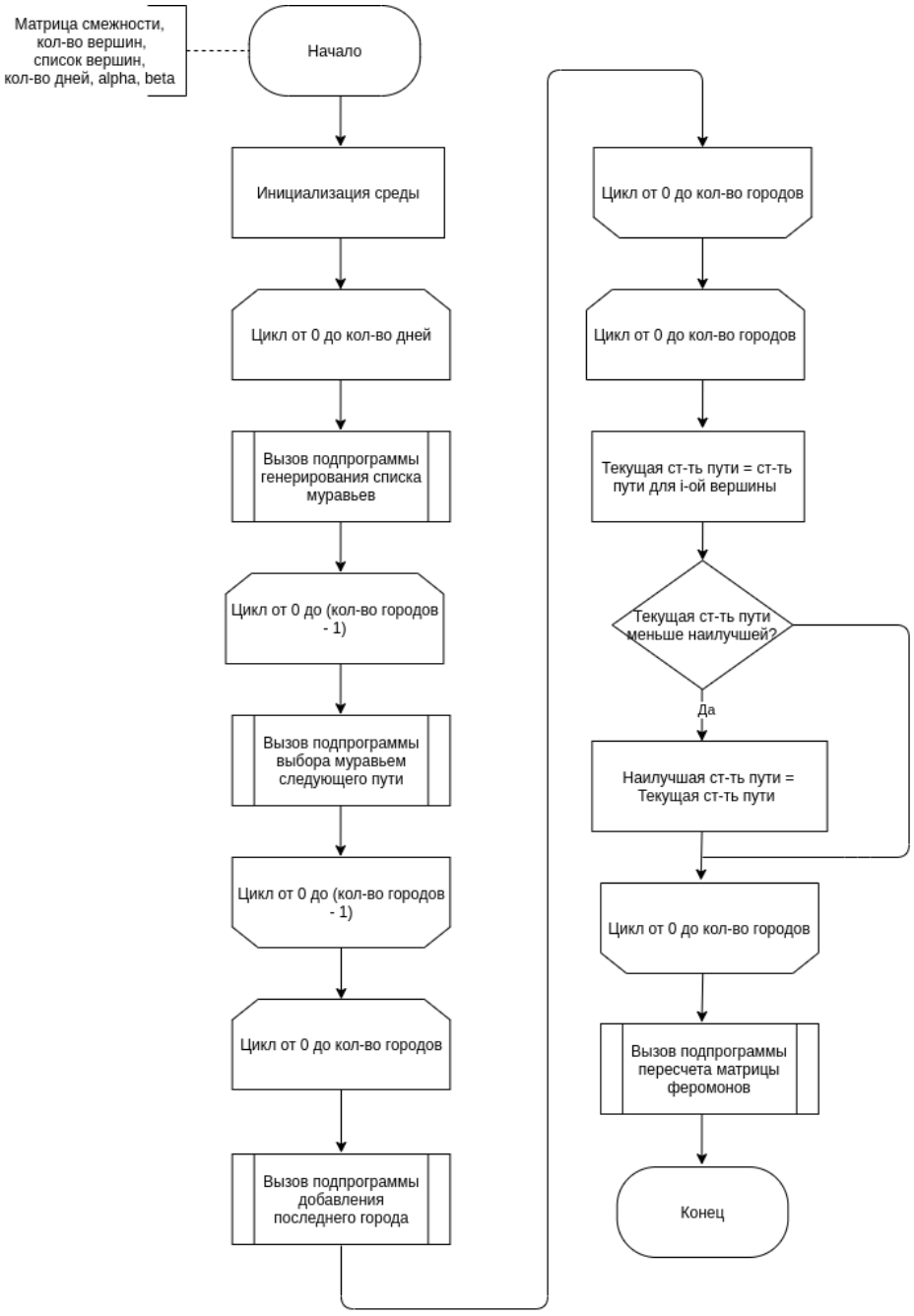

Рис. 2.2: Схема муравьиного алгоритма

3 | Технологическая часть

3.1 Выбор ЯП

3.2 Выбор языка программирования

В данной лабораторной работе использовался язык программирования - C [4], так как данный язык программирования предоставляет удобные библиотеки и инструменты для работы со структурами данных и обеспечивает хорошую производительность программного продукта. В качестве среды разработки я использовала Visual Studio Code [1]. Visual Studio Code подходит не только для Windows [2], но и для Linux [3], это причина, по которой я выбрала VS code, т.к. у меня установлена ОС Ubuntu 18.04.4 [5].

3.3 Сведения о модулях программы

Программа состоит из следующих модулей:

- main.c - главный файл программы, в котором располагается точка входа в программу;
- ant_algorithm.c - реализация муравьиного алгоритма;
- brute_force.c - реализация алгоритма полного перебора;
- matrix.c - модуль для работы с матрицей смежности;
- array.c - модуль для организации списка;
- city.c - модуль для организации списка городов;

- parser.c - модуль для представления результата в графическом виде.

На листингах 3.1-3.3 представлен основной код программы.

Листинг 3.1: Алгоритм полного перебора

```
int get_path_cost(array cities, int matrix[LEN][LEN])
{
  int cost = 0;

  for (int i = 0; i < cities.count - 1; i++)
    cost += matrix[cities.arr[i]][cities.arr[i + 1]];

  return cost;
}

array get_shortest_path(array cities, int matrix[LEN][LEN])
{
  array result[DEPTH_OF_RECURSION];
  array res_arr;

  int min_cost;
  int curr_cost;
  int index = 0;

  int routes_count = 0;

  del_elem(&cities, 0);
  add_elem(&res_arr, 0);
  get_routes(&cities, &res_arr, result, &routes_count);

  min_cost = get_path_cost(result[index], matrix);
  for (int i = 1; i < routes_count; i++)
  {
    curr_cost = get_path_cost(result[i], matrix);
    if (curr_cost < min_cost)
    {
      min_cost = curr_cost;
      index = i;
    }
```

```
35    }
36
37    return result[index];
38  }
39
40  void get_routes(array *cities, array *res_arr, array result
        [DEPTH_OF_RECURSION], int *count)
41  {
42    int elem;
43    array tmp;
44
45    if (!cities->count)
46    {
47      add_elem(res_arr, get_elem(*res_arr, 0));
48      result[*count] = *res_arr;
49      (*count)++;
50      del_elem(res_arr, res_arr->count - 1);
51    }
52
53    for (int i = 0; i < cities->count; i++)
54    {
55      elem = get_elem(*cities, i);
56      add_elem(res_arr, elem);
57
58      tmp = copy_arr(*cities);
59      del_elem(&tmp, i);
60
61      get_routes(&tmp, res_arr, result, count);
62
63      del_elem(res_arr, res_arr->count - 1);
64    }
65  }
```

Листинг 3.2: Функции для работы с муравьями

```
1  void generate_ants_array(ant ants[ANTS_MAX_COUNT], int
       count)
2  {
3    int elem;
4    for (int i = 0; i < count; i++)
5    {
```

```
6       ants[i].way.count = 0;
7       ants[i].route.count = 0;
8       elem = rand() % count;
9       add_elem(&(ants[i].way), elem);
10      for (int j = 0; j < count; j++)
11      {
12        if (j == elem)
13          continue;
14        add_elem(&(ants[i].route), j);
15      }
16    }
17  }
18
19  void choice_next_city(ant *ants, float matrix_pheromones[
        LEN][LEN], int matrix[LEN][LEN], float alpha, float beta
        )
20  {
21    float numerator = 0;
22    float denominator = 0;
23
24    float tao, reverse_cost;
25    int cost;
26
27    int city_curr = ants->way.arr[ants->way.count - 1];
28    for (int i = 0; i < ants->route.count; i++)
29    {
30      cost = matrix[city_curr][ants->route.arr[i]];
31      tao = matrix_pheromones[city_curr][ants->route.arr[i]];
32
33      if (!cost)
34        continue;
35
36      reverse_cost = 1.0 / cost;
37
38      denominator += powf(tao, alpha) + powf(reverse_cost,
            beta);
39    }
40
41    float p_array[LEN];
42    float sum = 0;
```

```
43    for (int i = 0; i < ants->route.count; i++)
44    {
45      cost = matrix[city_curr][ants->route.arr[i]];
46      tao = matrix_pheromones[city_curr][ants->route.arr[i]];
47
48      if (!cost)
49        continue;
50
51      reverse_cost = 1.0 / cost;
52      p_array[i] = (powf(tao, alpha) + powf(reverse_cost,
            beta)) / denominator;
53      sum += p_array[i];
54    }
55    float x = (float)rand() / RAND_MAX;
56    int index = 0;
57    while (x >= 0)
58    {
59      x -= p_array[index];
60      index++;
61    }
62    add_elem(&ants->way, get_elem(ants->route, index - 1));
63    del_elem(&ants->route, index - 1);
64  }
65
66  void ants_choose_way(ant ants[ANTS_MAX_COUNT], float
        matrix_pheromones[LEN][LEN], int matrix[LEN][LEN], int
        count, float alpha, float beta)
67  {
68    for (int i = 0; i < count; i++)
69      choice_next_city(&ants[i], matrix_pheromones, matrix,
           alpha, beta);
70  }
```

Листинг 3.3: Муравьиный алгоритм

```
1  int calculate_Q(int matrix[LEN][LEN], int count)
2  {
3    int Q = 0;
4
5    for (int i = 0; i < count; i++)
6      for (int j = 0; j < i; j++)
```

```
7         Q += matrix[i][j];
8
9     return Q * 2;
10  }
11
12  void add_pheromones(int matrix[LEN][LEN], float
        matrix_pheromones[LEN][LEN], int count, int Q, ant ants[
        ANTS_MAX_COUNT])
13  {
14    int city_first, city_second;
15    int curr_cost;
16    float delta_tao = 0;
17
18    for (int i = 0; i < count; i++)
19    {
20      curr_cost = get_path_cost(ants[i].way, matrix);
21
22      delta_tao += (float)Q / curr_cost;
23    }
24
25    for (int i = 0; i < count; i++)
26    {
27      for (int j = 0; j < ants[i].way.count - 1; j++)
28      {
29        city_first = ants[i].way.arr[j];
30        city_second = ants[i].way.arr[j + 1];
31        matrix_pheromones[city_first][city_second] =
              matrix_pheromones[city_first][city_second] +
              delta_tao;
32        matrix_pheromones[city_second][city_first] =
              matrix_pheromones[city_second][city_first] +
              delta_tao;
33      }
34    }
35  }
36
37  void correct_pheromones(float matrix_pheromones[LEN][LEN],
        int count)
38  {
39    for (int i = 0; i < count; i++)
```

```
40      for (int j = 0; j < i; j++)
41        if (matrix_pheromones[i][j] <= 0.1)
42          matrix_pheromones[i][j] = matrix_pheromones[j][i] =
                0.1;
43  }
44
45  void evaporation(float matrix_pheromones[LEN][LEN], int
        count, float p)
46  {
47    float tmp = 1 - p;
48    for (int i = 0; i < count; i++)
49      for (int j = 0; j < i; j++)
50      {
51        matrix_pheromones[i][j] = tmp * matrix_pheromones[i][
              j];
52        matrix_pheromones[j][i] = tmp * matrix_pheromones[j][
              i];
53      }
54  }
55  array ant_algorithm(int matrix[LEN][LEN], int count, array
        cities, int tmax, float p, float alpha, float beta)
56  {
57    int Q = calculate_Q(matrix, count);
58
59    array best_way = copy_arr(cities);
60    add_elem(&best_way, get_elem(best_way, 0));
61
62    int best_cost = get_path_cost(best_way, matrix);
63    int curr_cost = 0;
64
65    float matrix_pheromones[LEN][LEN];
66    fill_matrix(matrix_pheromones, count, PHEROMONE_MIN);
67
68    ant ants[ANTS_MAX_COUNT];
69
70    for (int t = 0; t < tmax; t++)
71    {
72      generate_ants_array(ants, count);
73
74      for (int i = 0; i < count - 1; i++)
```

```
75      {
76        ants_choose_way(ants, matrix_pheromones, matrix,
              count, alpha, beta);
77      }
78      for (int i = 0; i < count; i++)
79        add_elem(&ants[i].way, get_elem(ants[i].way, 0));
80
81      for (int i = 0; i < count; i++)
82      {
83        curr_cost = get_path_cost(ants[i].way, matrix);
84        if (curr_cost < best_cost)
85        {
86          best_cost = curr_cost;
87          best_way = copy_arr(ants[i].way);
88        }
89      }
90
91      evaporation(matrix_pheromones, count, p);
92      add_pheromones(matrix, matrix_pheromones, count, Q,
            ants);
93      correct_pheromones(matrix_pheromones, count);
94    }
95
96    return best_way;
97  }
```

3.4 Вывод

В данном разделе были разобраны листинги рис 3.1-3.2, показывающие работу алгоритмов для решения задачи коммивояжера.

4 | Экспериментальная часть

В данном разделе будет произведено сравнение двух алгоритмов:

1. полный перебор;
2. муравьиный алгоритм.

4.1 Сравнение времени работы муравьиного алгоритма и полного перебора

Для сравнения возьмем 8 массивов городов размерностью [3, 4, 5, 6, 7, 8, 9, 10]. Воспользуемся усреднением массового эксперимента.

Результат сравнения муравьиного алгоритма и полного перебора представлен на рис. 4.1.

По результатам эксперимента видно, что время испольнения муравьиного алгоритма значительно меньше, чем исполнение алгоритма полного перебора.

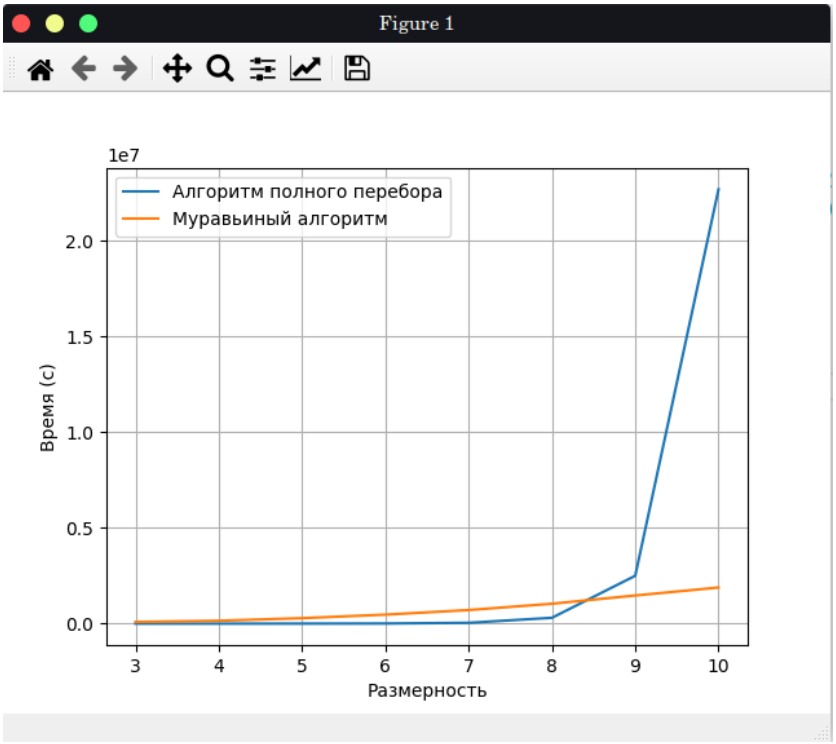

Рис. 4.1: Время работы муравьиного алгоритма и полного перебора

4.2 Результат работы программы

На рисунках 4.2 – 4.4 приведены результаты работы программы. Синим цветом показан путь, который нашел муравьиный алгоритм. Красным цветом показан путь, найденный полным перебором. На рисунке 4.3 показана ситуация, когда муравьиный алгоритм ошибся. Также на рисунке 4.4 показан вывод в консоль.

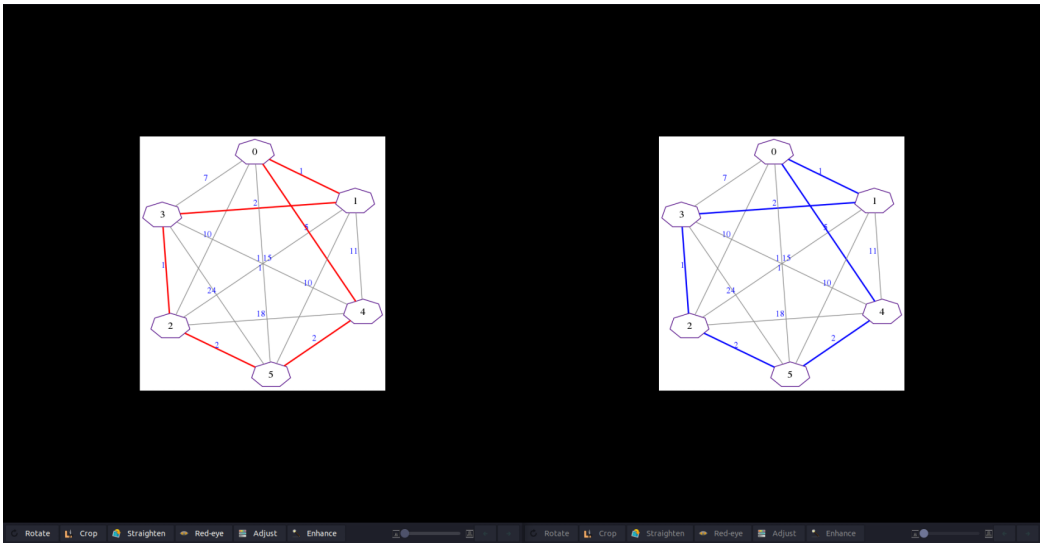

Рис. 4.2: Результат работы программы

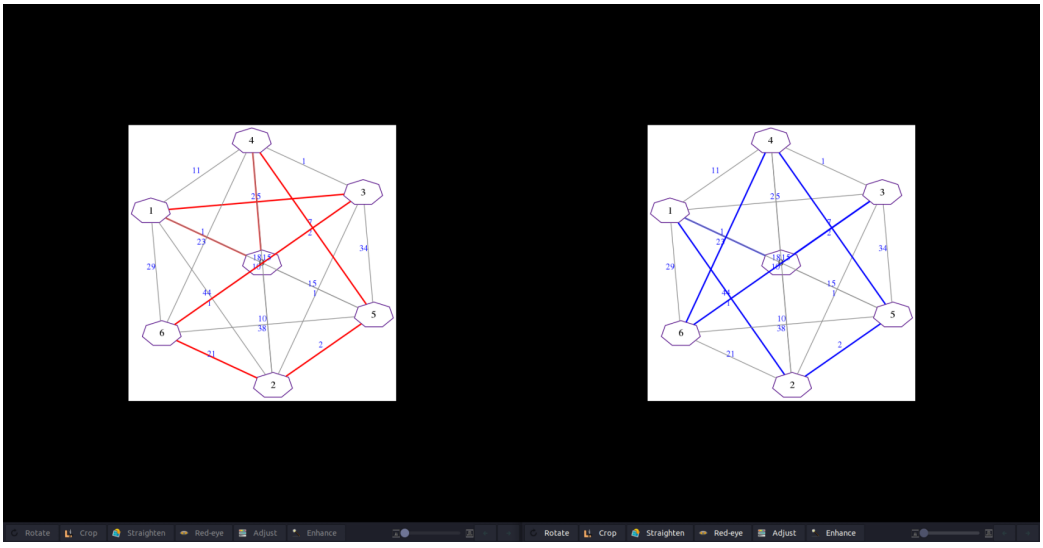

Рис. 4.3: Результат работы программы

4.3 Параметризация муравьиного алгоритма

В муравьином алгоритме вычисления производятся на основе настраиваемых параметров.

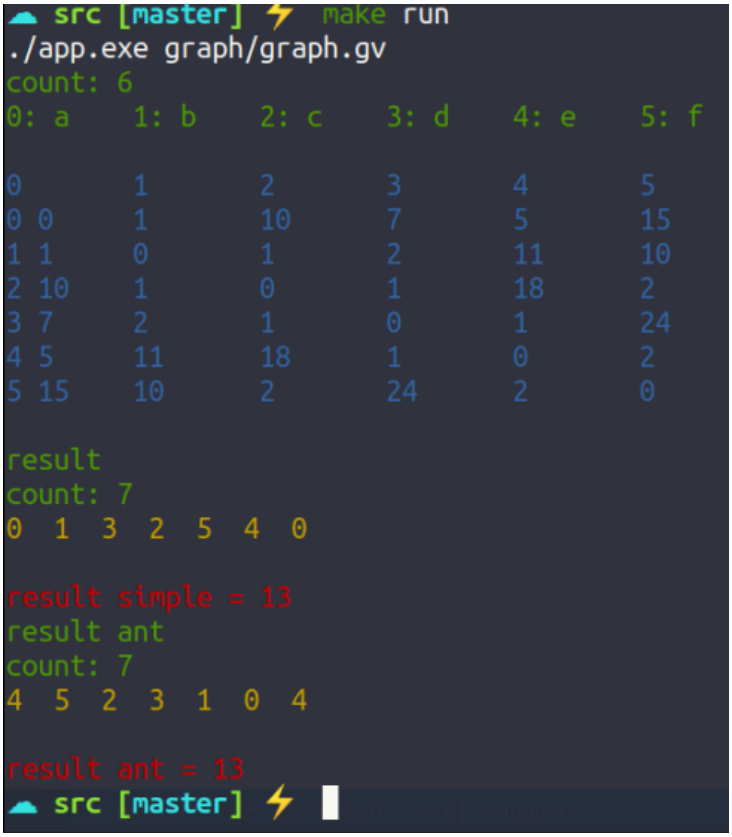

Рис. 4.4: Результат работы программы

Рассмотрим матрицу смежностей размерностью 10×10 (4.1)

Таблица 4.1: Матрица смежностей

0	0	1	2	3	4	5	6	7	8	9
0	0	1790	200	1900	63	1659	1820	1395	2382	649
1	1790	0	1573	2435	1515	714	892	2193	1590	1003
2	200	1573	0	833	392	2404	962	902	141	1123
3	1900	2435	833	0	2283	1652	2362	2262	1512	2166
4	63	1515	392	2283	0	1322	290	1305	2100	969
5	1659	714	2404	1652	1322	0	256	78	2236	2041
6	1820	892	962	2362	290	256	0	1180	1547	1279
7	1395	2193	902	2262	1305	78	1180	0	1640	1161
8	2382	1590	141	1512	2100	2236	1547	1640	0	2212
9	649	1003	1123	2166	969	2041	1279	1161	2212	0

Параметризация метода решения задачи коммивояжера на основании муравьиного алгоритма проводилась для матрицы с элементами в диапозоне [0, 2500]. Количество дней было равно 50. Полный перебор определил оптимальную длину пути 6986. Столбец "результат" отвечает за результат работы муравьиного алгоритма. Столбец "разница" отвечает за разницу с оптимальной длиной.

4.4 Вывод

На основе проведенной параметризации (таблицы 4.2–4.5) для матрицы смежности приведенной в таблице (4.1) рекомендуется использовать ($\alpha = 0.5, \beta = 0.5, \rho =$ любое). При этих параметрах, количество правильно найденных оптимальных путей составило 8 единиц.

Таблица 4.2: Таблица коэффициентов.Часть 1

α	β	p	Результат	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

Таблица 4.3: Таблица коэффициентов.Часть 2

α	β	p	Результат	Разница
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

Таблица 4.4: Таблица коэффициентов.Часть 3

α	β	p	Результат	Разница
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
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

Таблица 4.5: Таблица коэффициентов.Часть 4

α	β	p	Результат	Разница
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

Заключение

В данной лабораторной работе были рассмотрены основополагающие материалы которые в дальнейшем потребовались при реализации алгоритма полного перебора и муравьиного алгоритма. Были рассмотрены схемы (рис. 2.1–2.2) для решения задачи коммивояжера. Также были разобраны листинги 3.1-3.2, показывающие работу, описанных выше, алгоритмов и были приведены рисунки 4.2 - 4.4, показывающий работу алгоритмов. Был произведен сравнительный анализ рис. 4.1.

В рамках выполнения работы решены следующие задачи:

1. изучены два алгоритма для решения задачи коммивояжера;
2. применены изученные основы для реализации двух алгоритмов;
3. получены практические навыки;
4. проведена параметризация муравьиного алгоритма;
5. проведен сравнительный анализ скорости работы реализованных алгоритмов;
6. описан и обоснован выбор язык программирования, для решения поставленной задачи.

Литература

[1] Visual Studio Code [Электронный ресурс], режим доступа: https://code.visualstudio.com/ (дата обращения: 02.10.2020)

[2] Windows [Электронный ресурс], режим доступа:https://www.microsoft.com/ru-ru/windows (дата обращения: 02.10.2020)

[3] Linux [Электронный ресурс], режим доступа:https://www.linux.org.ru/ (дата обращения: 02.10.2020)

[4] Справочник по языку C [Электронный ресурс], режим доступа:https://docs.microsoft.com/ru-ru/cpp/c-language/c-language-reference?view=msvc-160 (дата обращения: 25.12.2020)

[5] Ubuntu 18.04 [Электронный ресурс], режим доступа:https://releases.ubuntu.com/18.04/ (дата обращения: 02.10.2020)