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**

ОТЧЕТ

по лабораторной работе № 6

Название: Муравьиный алгоритм

Дисциплина: Анализ алгоритмов

Студент ИУ7-52Б _____ Короткая В. М.

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Волкова Л.Л.

(Подпись, дата) (И.О. Фамилия)

Москва, 2021

Содержание

Введение

Муравья нельзя назвать сообразительным. Отдельный муравей не в состоянии принять ни малейшего решения. Дело в том, что он устроен крайне примитивно: все его действия сводятся к элементарным реакциям на окружающую обстановку и своих собратьев. Муравей не способен анализировать, делать выводы и искать решения.

Эти факты, однако, никак не согласуются с успешностью муравьев как вида. Они существуют на планете более 100 миллионов лет, строят огромные жилища, обеспечивают их всем необходимым и даже ведут настоящие войны. В сравнении с полной беспомощностью отдельных особей, достижения муравьев кажутся немыслимыми.

Добиться таких успехов муравьи способны благодаря своей социальности. Они живут только в коллективах – колониях. Все муравьи колонии формируют так называемый роевой интеллект. Особи, составляющие колонию, не должны быть умными: они должны лишь взаимодействовать по определенным – крайне простым – правилам, и тогда колония целиком будет эффективна.

В колонии нет доминирующих особей, нет начальников и подчиненных, нет лидеров, которые раздают указания и координируют действия. Колония является полностью самоорганизующейся. Каждый из муравьев обладает информацией только о локальной обстановке, не один из них не имеет представления обо всей ситуации в целом – только о том, что узнал сам или от своих сородичей, явно или неявно. На неявных взаимодействиях муравьев, называемых стигмергией, основаны механизмы поиска кратчайшего пути от муравейника до источника пищи.

Каждый раз проходя от муравейника до пищи и обратно, муравьи оставляют за собой дорожку феромонов. Другие муравьи, почувствовав такие следы на земле, будут инстинктивно устремляться к нему. Поскольку эти муравьи тоже оставляют за собой дорожки феромонов, то чем больше муравьев проходит по определенному пути, тем более привлекательным он становится для их сородичей. При этом, чем короче путь до источника пищи, тем меньше времени требуется муравьям на него – а следовательно, тем быстрее оставленные на нем следы становятся заметными.

В 1992 году в своей диссертации Марко Дориго (Marco Dorigo) предложил заимствовать описанный природный механизм для решения задач оптимизации. Имитируя поведение колонии муравьев в природе, муравьиные алгоритмы используют многоагентные системы, агенты которых функционируют по крайне простым правилам. Они крайне эффективны при решении сложных комбинаторных задач – таких, например, как задача коммивояжера, первая из решенных с использованием данного типа алгоритмов.

Целью данной работы является изучение и реализация двух алгоритмов:

- полный перебор;
- муравьиный алгоритм.

В рамках выполнения работы необходимо решить следующие задачи:

- изучить два, описанных выше алгоритма для решения задачи коммивояжера;
- применить изученные знания для реализации двух алгоритмов;
- составить схемы алгоритмов;
- провести параметризацию муравьиного алгоритма;
- провести сравнительный анализ скорости работы реализованных алгоритмов;
- выбрать и обосновать выбор языка программирования, для решения поставленной задачи.

1. Аналитическая часть

В данном разделе будут рассмортенно формальное описание алгоритмов.

1.1. Задача коммивояжера

В 19-м и 20-м веке по городам ездили коммивояжёры (сейчас их называют «торговые представители»). Они ходили по домам и предлагали людям купить разные товары. Тактика была такой: коммивояжёр приезжал в город, обходил большинство домов и отправлялся в следующий город. Города были небольшими, поэтому обойти всё было вполне реально.

Чем больше городов посетит коммивояжёр, тем больше домов он сможет обойти и больше заработать с продаж.

В задаче коммивояжера рассматривается n 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. Решение полным перебором

Эту задачу возможно решить полным перебором т. е. разобрать все возможные варианты и выбрать оптимальный. Но проблема такого решения в том, что с увилечением количества городов, время выполнения будет расти.

Хотя такой подход и гарантирует точное решение задачи, уже при небольшом числе городов решение задачи за допустимое время не возможно.

1.3. Решение муравьиным алгоритмом

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.

Каждый муравей хранит в памяти список пройденных им узлов. Этот список называют списком запретов (tabu list) или просто памятью муравья. Выбирая узел для следующего шага, муравей «помнит» об уже пройденных узлах и не рассматривает их в качестве возможных для перехода. На каждом шаге список запретов пополняется новым узлом, а перед новой итерацией алгоритма – то есть перед тем, как муравей вновь проходит путь – он опустошается.

Кроме списка запретов, при выборе узла для перехода муравей руководствуется «привлекательностью» ребер, которые он может пройти. Она зависит, во-первых, от расстояния между узлами (то есть от веса ребра), а во-вторых, от следов феромонов, оставленных на ребре прошедшими по нему ранее муравьями. Естественно, что в отличие от весов ребер, которые являются константными, следы феромонов обновляются на каждой итерации алгоритма: как и в природе, со временем следы испаряются, а проходящие муравьи, напротив, усиливают их.

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.

Вывод

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

Входными данными реализуемого ПО являются:

- размерность матрицы - натуральное число;
- матрица - целочисленная, описывает граф.

Выходными данными реализуемого ПО являеться результат алгоритмов т. е.:

- последовательность вершин - для полного перебора;
- последовательность вершин - для муравьиного алгоритма.

2. Конструкторская часть

В данном разделе представлены схемы алгоритмов. Так же будут описаны пользовательские структуры данных, приведены структура ПО и классы эквивалентности для тестирования реализуемого ПО.

2.1. Схемы алгоритмов

На рис. 2.1 показана схема алгоритма полного перебора.

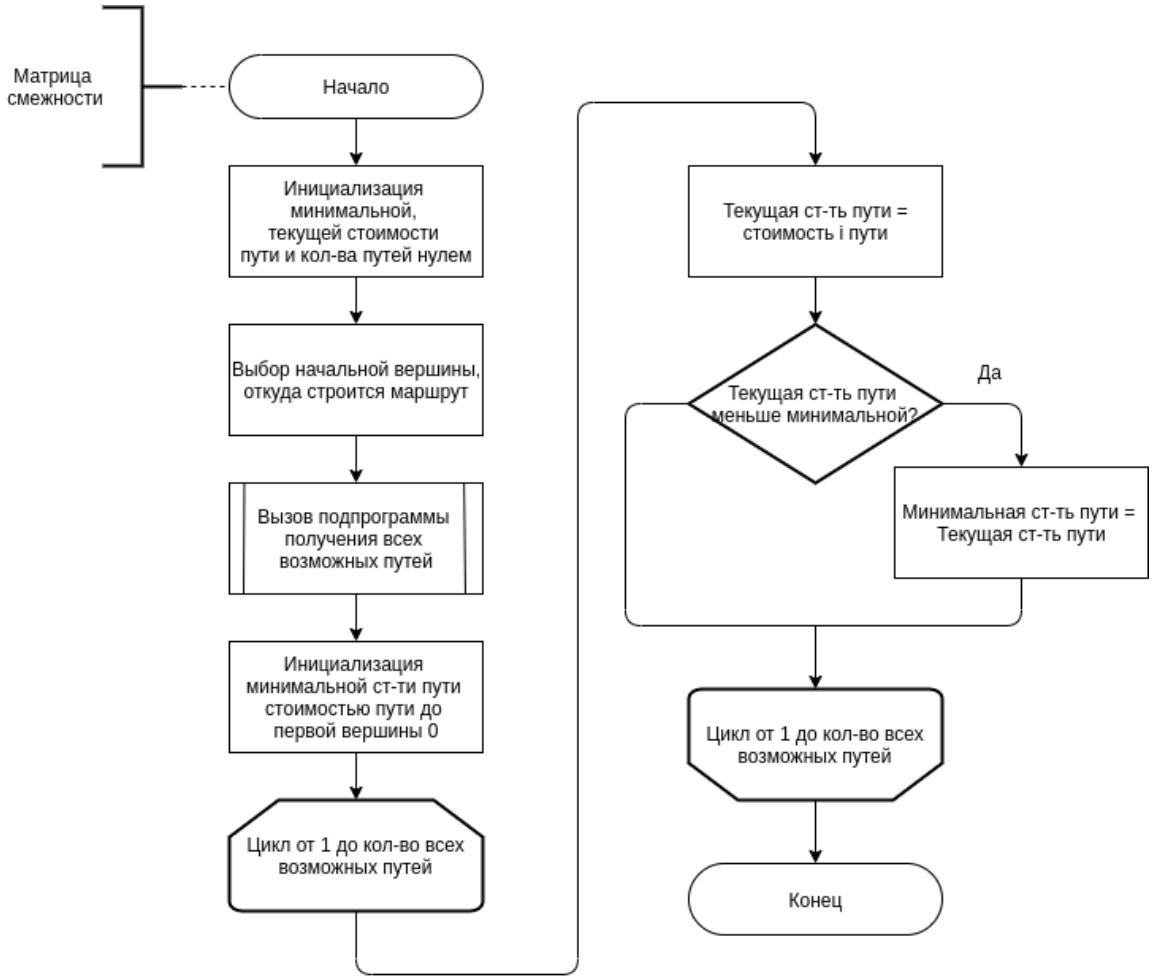

Рис. 2.1: Схема алгоритма полного перебора

На рис. 2.2 показана схема муравьиного алгоритма.

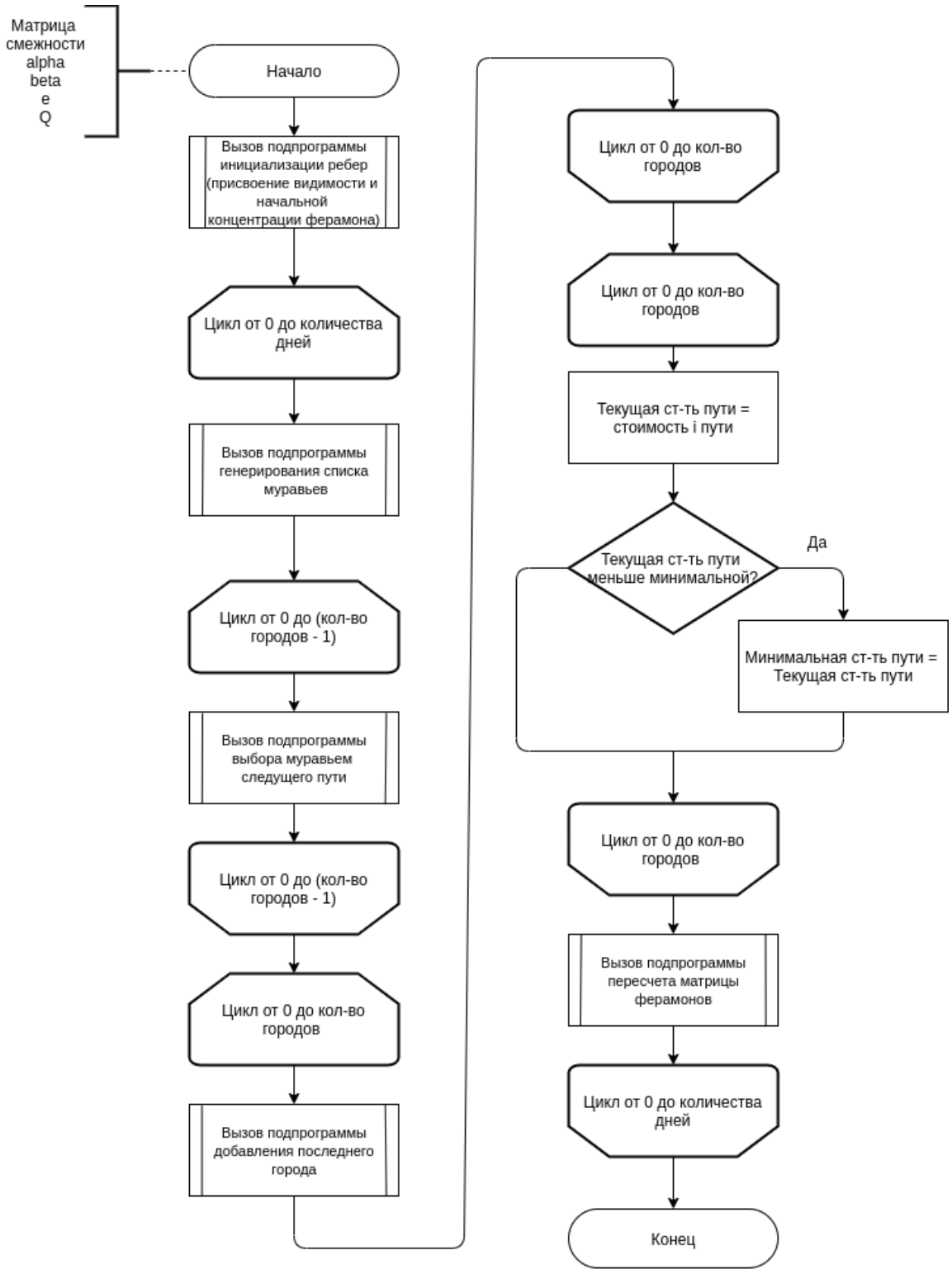

Рис. 2.2: Схема муравьиного алгоритма

2.2. Описание структур данных

Для реализации данных алгоритмов, введем некоторые типы данных:

- ant - тип данных описывающий муравья:
  - way - массив посещенных вершин;
  - route - массив не посищенных вершин.
- antColony - тип данных описывающий колонию муравьев:
  - graph - граф, описанный матрицей смежности;
  - ph - матрица, описывающая феромон;
  - alpha - параметр α;
  - beta - параметр β;
  - Q - параметр Q;
  - p - параметр ρ.

2.3. Классы эквивалентности

В рамках данной лабороторной можно выделить следующие классы эквивалентности:

- входными данными является ациклический ориентированный взвешенный граф;
- входными данными является циклический ориентированный взвешенный граф.

Для проверки корректности работы будет проведено функциональное тестирование согласно классам эквивалентности.

Вывод

На основе теоритических данных, полученных из аналитического раздела, были построены схемы реализируемых алгоритмов.

Так же, было приведено описание вводимых типов данных.

3. Технологическая часть

В данном разделе приведены средства реализации, требования к ПО и листинги кода.

3.1. Средства реализации

В качестве языка программирования был выбран c++. Данный язык знаком и предостовляет все необходимые ресурсы. В качестве среды разработки я использовала Visual Studio Code, т.к. считаю его достаточно удобным и легким. Visual Studio Code подходит не только для Windows, но и для Linux, это еще одна причина, по которой я выбрала VS code, т.к. у меня установлена ОС fedora 34.

3.2. Сведенья о модулях программы

- main.cpp - файл, содержащий точку входа в программу;
- bruteforce.cpp - файл, содержащий реализацию алгоритма полного перебора;
- ant.cpp - файл, содержащий реализацию поведения муравья;
- antalgorithm.cpp - файл, содержащий реализацию муравьиного алгоритма;
- array.cpp - файл, содержащий реализацию работы с массивом;
- matrix.cpp - файл, содержащий реализацию работы с матрицей.

3.3. Реализация алгоритмов

В листингах 3.1 - 3.3 приведены реализации алгоритмов на ЯП C++.

Листинг 3.1: Функция релизующая полный перебор.

```
1 array get_shortest_path(array cities, int matrix[LEN][LEN])
2 {
3         array result[DEPTH_OF_RECURSION];
4         array res_arr;
5
```

```
6           int min_cost;
7           int curr_cost;
8           int index = 0;
9
10          int routes_count = 0;
11
12          del_elem(&cities, 0);
13          add_elem(&res_arr, 0);
14          get_routes(&cities, &res_arr, result, &routes_count);
15
16          min_cost = get_path_cost(result[index], matrix);
17          for (int i = 1; i < routes_count; i++)
18          {
19                  curr_cost = get_path_cost(result[i], matrix);
20                  if (curr_cost < min_cost)
21                  {
22                          min_cost = curr_cost;
23                          index = i;
24                  }
25          }
26
27          return result[index];
28  }
```

Листинг 3.2: Функция реализующая муравьинный алгоритм

```
1  array ant_algorithm(int matrix[LEN][LEN], int count,
2   array cities, int tmax, float p, float alpha, float beta)
3  {
4      int Q = calculate_Q(matrix, count);
5      array best_way = copy_arr(cities);
6      add_elem(&best_way, get_elem(best_way, 0));
7
8      int best_cost = get_path_cost(best_way, matrix);
9      int curr_cost = 0;
10
```

```
11      float matrix_pheromones[LEN][LEN];
12      fill_matrix(matrix_pheromones, count, PHEROMONE_MIN);
13
14      ant ants[ANTS_MAX_COUNT];
15      for (int t = 0; t < tmax; t++)
16      {
17          generate_ants_array(ants, count);
18          for (int i = 0; i < count - 1; i++)
19          {
20              ants_choose_way(ants, matrix_pheromones,
21                                  matrix, count, alpha, beta);
22          }
23          for (int i = 0; i < count; i++)
24              add_elem(&ants[i].way, get_elem(ants[i].way, 0));
25          for (int i = 0; i < count; i++)
26          {
27              curr_cost = get_path_cost(ants[i].way, matrix);
28              if (curr_cost < best_cost)
29              {
30                  best_cost = curr_cost;
31                  best_way = copy_arr(ants[i].way);
32              }
33          }
34          evaporation(matrix_pheromones, count, p);
35          add_pheromones(matrix, matrix_pheromones,
36                                          count, Q, ants);
37          correct_pheromones(matrix_pheromones, count);
38      }
39      return best_way;
40  }
```

Листинг 3.3: Функция вычисляющая Q.

```
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  }
```

Листинг 3.4: Функция реализации добавления ферамона.

```
1   void add_pheromones(int matrix[LEN][LEN],
2     float matrix_pheromones[LEN][LEN], int count, int Q,
3                                 ant ants[ANTS_MAX_COUNT])
4   {
5       int city_first, city_second;
6       int curr_cost;
7       float delta_tao = 0;
8
9       for (int i = 0; i < count; i++)
10      {
11          curr_cost = get_path_cost(ants[i].way, matrix);
12          delta_tao += (float)Q / curr_cost;
13      }
14
15      for (int i = 0; i < count; i++)
16      {
17          for (int j = 0; j < ants[i].way.count - 1; j++)
18          {
19              city_first = ants[i].way.arr[j];
20              city_second = ants[i].way.arr[j + 1];
21              matrix_pheromones[city_first][city_second] =
22               matrix_pheromones[city_first][city_second]
23                                                  + delta_tao;
24              matrix_pheromones[city_second][city_first] =
25               matrix_pheromones[city_second][city_first]
26                                                  + delta_tao;
```

```
27          }
28      }
29  }
```

Листинг 3.5: Функция создания муравьев.

```
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 }
```

Листинг 3.8: Функция реализующая выбор следующего города для муравьем.

```
1  void choice_next_city(ant *ants,
2   float matrix_pheromones[LEN][LEN], int matrix[LEN][LEN],
3                                      float alpha, float beta)
4  {
5      float numerator = 0;
6      float denominator = 0;
7      float tao, reverse_cost;
8      int cost;
9
10     int city_curr = ants->way.arr[ants->way.count - 1];
11     for (int i = 0; i < ants->route.count; i++)
```

```
12      {
13          cost = matrix[city_curr][ants->route.arr[i]];
14          tao = matrix_pheromones[city_curr][ants->route.arr[i]]
15
16          if (!cost)
17                      continue;
18
19          reverse_cost = 1.0 / cost;
20
21          denominator += powf(tao, alpha) +
22                         powf(reverse_cost, beta);
23      }
24
25      float p_array[LEN] = {0};
26      float sum = 0;
27      for (int i = 0; i < ants->route.count; i++)
28      {
29          cost = matrix[city_curr][ants->route.arr[i]];
30          tao = matrix_pheromones[city_curr][ants->route.arr[i]]
31
32          reverse_cost = 1.0 / cost;
33
34          p_array[i] = (powf(tao, alpha) +
35                  powf(reverse_cost, beta)) / denominator;
36      }
37
38      float x = (float)rand() / RAND_MAX;
39      int index = 0;
40      while (x >= 0)
41      {
42          x -= p_array[index];
43          index++;
44      }
45
```

```
46      add_elem(&ants->way, get_elem(ants->route, index - 1));
47      del_elem(&ants->route, index - 1);
48  }
```

Листинг 3.9: Функция реализующая выбор пути для муравьем

```
1  void ants_choose_way(ant ants[ANTS_MAX_COUNT],
2  float matrix_pheromones[LEN][LEN], int matrix[LEN][LEN],
3   int count, float alpha, float beta)
4  {
5      for (int i = 0; i < count; i++)
6      choice_next_city(&ants[i], matrix_pheromones,
7                                  matrix, alpha, beta);
8  }
```

3.4. Тестирование

В таблице 3.1 приведены функциональные тесты для алгоритмов сортировки.

Таблица 3.1: Таблица тестов

Первая матрица	Ожидаемый результат	Жадный	Муравьиный
$\begin{bmatrix} 0 & 3 & 1 & 6 & 8 \\ 3 & 0 & 4 & 1 & 0 \\ 1 & 4 & 0 & 5 & 0 \\ 6 & 1 & 5 & 6 & 1 \\ 8 & 0 & 0 & 1 & 1 \end{bmatrix}$	15	15	15
$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 10 & 0 & 35 & 25 \\ 15 & 35 & 0 & 30 \\ 20 & 25 & 30 & 0 \end{bmatrix}$	80	80	80

При проведении функционального тестирования, полученные результаты работы программы совпали с ожидаемыми. Таким образом, функциональное тестирование пройдено успешно.

Вывод

В данном разделе были реализованны вышеописанные алгоритмы. Было разработано ПО, удовлетворяющее предьявляемым требованиям. Так же были представлены соответствующие листинги с кодом программы. А так же проведено тестирование разработанного ПО.

4. Исследовательская часть

В данном разделе будет произведено измерение временных характеристик. А так же будет выполнена параметризация алгоритма.

4.1. Технические характеристики

Технические характеристики устройства на котором выполнялось исследование:

- процессор Intel® Core™ i5-10210U CPU @ 1.60GHz × 8;
- память 15.3 GiB;
- операционная система Fedora 34 (Workstation Edition) 64-bit.

4.2. Временные характеристики

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.

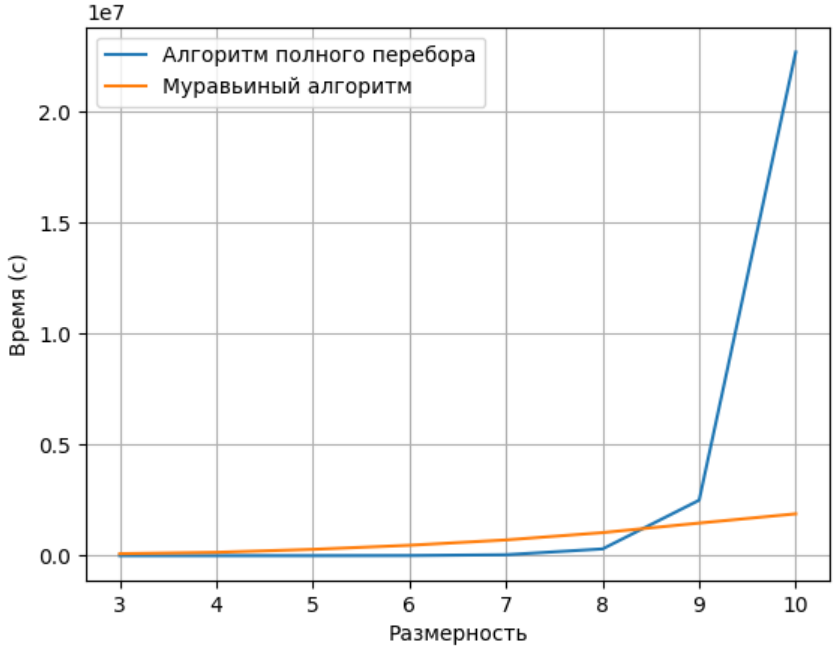

Рис. 4.1: Время работы муравьиного алгоритма и полного перебора

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

4.3. Параметризация муравьиного алгоритма

В муравьином алгоритме вычисления производятся на основе настраиваемых параметров.

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

Таблица 4.2: Таблица коэффициентов.(Полную таблицу см. Приложение)

α	β	p	Результат	Разница
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	0.7	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
0.3	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
0.6	0.4	1	6986	0
0.9	0.1	0.9	7376	390
1	0	0.8	7874	888
1	0	0.9	7446	460
1	0	1	8119	1133

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.

Вывод

На основе проведенной параметризации (таблицы 4.3–4.2) 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.

Заключение

В данной лабораторной работе были рассмотрены основополагающие материалы которые в дальнейшем потребовались при реализации алгоритма полного перебора и муравьиного алгоритма. Были рассмотрены схемы для решения задачи коммивояжера. Также были разобраны листинги , показывающие работу, описанных выше алгоритмов.. Был произведен сравнительный анализ.

В рамках выполнения работы решены следующие задачи:

- изучины два, описанных выше алгоритма для решения задачи коммивояжера;
- составлины схемы алгоритмов;
- проведены параметризацию муравьиного алгоритма;
- проведены сравнительный анализ скорости работы реализованных алгоритмов;

Список литературы

1. Дж. Макконнел. Анализ алгоритмов. Активный обучающий подход. – М.: Техносфера, 2017. – 267с.

2. Основы программирования на языках Си и C++ для начинающих[Электронный ресурс]. Режим доступа: http://cppstudio.com/ (дата обращения 10.10.2021)

3. LINUX.ORG.RU - Русскоязычная информация о ОС Linux[Электронный ресурс] Режим доступа://www.linux.org.ru/(дата обращения 25.10.2021)

4. Штовба С. Д. Муравьиные алгоритмы, Exponenta Pro. Математика в приложениях. 2004. № 4

5. Задача коммивояжера[Электронный ресурс] - режим доступа http://mech.math.msu.su/ shvetz/54/inf/perl-problems/chCommisVoyageur.xhtml (дата обращения 25.10.2021)

Приложение

Таблица 4.3: 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

Таблица 4.4: Таблица коэффициентов.Часть 2

α	β	p	Результат	Разница
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

Таблица 4.5: Таблица коэффициентов.Часть 3

α	β	p	Результат	Разница
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
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