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 «ФУНДАМЕНТАЛЬНЫЕ НАУКИ»

КАФЕДРА «ПРИКЛАДНАЯ МАТЕМАТИКА»

Лабораторная работа № 5

по дисциплине «Типы и структуры данных»

Тема Задача коммивояжёра

Студент Лямин И.С.

Группа ФН12-31Б

Преподаватели Волкова Л.Л.

Москва, 2024

Содержание

ВВЕДЕНИЕ

Целью работы является описание и реализация алгоритма для решения задачи коммивояжёра с помощью **муравьиного алгоритма** и **муравьиного алгоритма с элитными муравьями**. Осуществление поставленной цели требует выполнения следующих задач:

1) описание всех нужных понятий для реализации алгоритмов;
2) реализация алгоритмов;
3) тестирование полученной реализации;
4) проведение исследования с целью выявления наилучших параметров для решения задачи коммивояжёра на определённом кластере данных.

1 Аналитическая часть

1.1 Алгоритм муравья

Муравьиный алгоритм — это метаэвристический алгоритм для решения задач комбинаторной оптимизации, таких как задача коммивояжёра. Основной принцип работы алгоритма основан на моделировании поведения муравьёв, которые оставляют феромоны на пути между узлами, обозначая тем самым предпочтительные маршруты.

Муравьиный алгоритм с элитными муравьями — данный алгоритм является модификацией предыдущего. Его отличие заключается в добавлении дополнительного феромона на рёбра/грани, входящие в наилучшие маршруты. Этот феромон добавляется в фазе "ночи", когда обновляется весь феромон.

1.2 Основные понятия

— **Феромоны**: Муравьи оставляют на рёбрах графа вещество — феромон (обозначается как τ), уровень которого влияет на вероятность выбора данного пути.

— **Эвристическая информация**: Дополнительные данные о предпочтительности перехода, такие как обратное расстояние между двумя городами: $\eta_{ij} = \frac{1}{d_{ij}}$, где d_{ij} — расстояние между городами i и j.

— **Испарение феромонов**: Со временем уровень феромонов на рёбрах уменьшается, что позволяет алгоритму избегать зацикливания на субоптимальных решениях.

— **Комбинированная вероятность**: Выбор следующего города для посещения основан на сочетании уровня феромонов и эвристической информации.

1.3 Алгоритм работы

Алгоритм работает по описанию ниже.

1) **Инициализация**:
   — задаётся начальный уровень феромонов τ_{ij} для всех рёбер графа;
   — определяются параметры алгоритма — α, β, ρ и количество муравьёв.

2) **Построение маршрута**. Каждый муравей строит маршрут, постепенно посещая все города. Выбор следующего города j осуществляется на основе вероятности:

$$P_{ij} = \begin{cases} \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{k \in \text{непосещённые}} \tau_{ik}^{\alpha} \cdot \eta_{ik}^{\beta}}, & j \in \text{непосещённые} \\ 0 \end{cases} \tag{1.1}$$

где τ_{ij} — уровень феромонов на ребре ij, η_{ij} — эвристическая информация, α и β — параметры, управляющие влиянием феромонов и эвристической информации соответственно.

3) **Обновление феромонов**: После того как все муравьи завершили построение маршрутов, уровни феромонов обновляются:

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

где ρ — коэффициент испарения феромонов, $\Delta\tau_{ij}$ — добавка феромонов на основании качества маршрутов, использующих ребро ij.

Важно, что уровень феромонов на ребре никогда не опускается ниже предопрелённой константы $\varepsilon > 0$, тем самым обеспечивая вероятность выбора ребра, даже если оно находится на неразработанном маршруте.

Добавка феромонов вычисляется следующим образом:

$$\Delta\tau_{ij} = \sum_{k} \frac{Q}{L_k}, \tag{1.3}$$

где Q — параметр алгоритма, а L_k — длина маршрута k-го муравья.

В модернизации алгоритма выделяется дополнительный феромон на рёбрах, которые входят в лучшие маршруты.

4) **Повторение**: Шаги построения маршрутов и обновления феромонов повторяются, пока не будет достигнут критерий остановки (например, заданное число итераций(дней)).

2 Конструкторская часть

2.1 Используемые структуры данных и классы

1) **Класс** `Matrix` обеспечивает хранение и управление матрицами различных типов данных. Основные методы класса:

— `print` — вывод матрицы на экран;

— `zero` — заполнение матрицы нулевыми значениями;

2) **Основные переменные**:

— `D` — матрица расстояний между узлами графа;

— `T` — матрица феромонов;

— `Q` — квота феромона;

— `N` — количество узлов (городов) графа и количество муравьёв.

2.2 Инициализация

— Чтение входных данных из файла. Файл содержит описание графа, включая количество узлов и матрицу расстояний `D`.

— Заполнение начальной матрицы феромонов `T`. Феромоны инициализируются минимальными значениями для всех рёбер.

— Установка параметров алгоритма, таких как `a` (влияние расстояния), `b` (влияние феромона), и `q` (коэффициент испарения феромонов).

2.3 Основной цикл работы алгоритма

Алгоритм выполняется в течение заданного количества итераций (дней). Каждая итерация включает шаги, приведённые ниже:

Построение маршрутов муравьями

— Каждый муравей начинает маршрут из случайного узла графа(каждый муравей из разного узла).

— На каждом шаге муравей выбирает следующий узел на основе вероятностей, которые вычисляются с учётом:

- текущего уровня феромонов на рёбрах;
- обратной величины расстояния до следующего узла.

Обновление локальной матрицы феромонов

— После завершения маршрута каждого муравья обновляется локальная матрица феромонов на основе длины маршрута.

— Количество феромона, добавляемое на ребро, обратно пропорционально длине маршрута.

Обновление глобальной матрицы феромонов

— По завершении итерации выполняется глобальное обновление феромонов.
— Феромоны на каждом ребре испаряются с коэффициентом q.
— Значение τ_{ij} ограничивается минимальным уровнем, чтобы предотвратить полное исчезновение феромонов.

Выбор лучших маршрутов

— По окончании каждой итерации сохраняется маршрут с минимальной длиной.
— Если используется режим с элитными муравьями, феромоны дополнительно обновляются с учётом лучших маршрутов за всю историю работы алгоритма.

2.4 Функции

Функция `calculate_median`

Вычисляет медиану заданного вектора целых чисел.
— Входные данные: вектор `v`.
— Алгоритм:
    1) сортирует элементы вектора;
    2) если количество элементов нечётное, возвращает центральный элемент;
    3) если чётное, возвращает среднее арифметическое двух центральных элементов.
— Возвращаемое значение: медиана вектора.

Функция `calculate_mean`

Вычисляет среднее арифметическое элементов вектора.
— Входные данные: вектор `v`.
— Алгоритм:
    1) суммирует все элементы вектора;
    2) делит сумму на количество элементов.
— Возвращаемое значение: среднее арифметическое.

Функция `show_route`

Выводит маршрут, пройденный муравьём, и длины рёбер.

— Входные данные: указатель на вектор `nodes`, содержащий последовательность узлов маршрута.

— Алгоритм:

1) для каждой пары последовательных узлов выводит их и длину ребра между ними;

2) если включён режим цикла, также выводит ребро от последнего узла к первому.

— Вывод: маршрут и его длины.

Функция `get_length`

Вычисляет длину маршрута.

— Входные данные: указатель на вектор `nodes`.

— Алгоритм:

1) суммирует длины всех рёбер маршрута;

2) если включён режим цикла, добавляет длину ребра от последнего узла к первому.

— Возвращаемое значение: длина маршрута.

Функция `count_Q`

Вычисляет общее количество феромона `Q`, используемого для обновлений.

— Алгоритм:

1) суммирует все расстояния в матрице `D`;

2) делит сумму на N (или $N - 1$, в зависимости от режима).

— Возвращаемое значение: значение `Q`.

Функция `upgrade_pheromone`

Обновляет матрицу феромонов на основе лучших маршрутов(используется только в модернизированном алгоритме).

— Входные данные: указатель на вектор лучших маршрутов `routes`, длина лучшего маршрута `L_best`.

— Алгоритм:

– для каждого маршрута из `routes` добавляет к соответствующим рёбрам феромон, пропорциональный Q/L_best.

3 Технологическая часть

3.1 Выбор средств реализации

Для программной реализации алгоритма использовалась среда разработки Visual Studio 2022, язык программирования, на котором была выполнена реализации алгоритмов — C++. Для компиляции кода использовался компилятор MSVC. Исследование проводилось на ноутбуке (64-разрядная операционная система, процессор x64, частота процессора 3.1 ГГц, оперативная память 16 ГБ)

3.2 Реализация алгоритмов

В листинге 3.1 представлена программная реализация описанных алгоритмов.

Листинг 3.1 — Программная реализация алгоритма и всех вспомогательных функций

```
#include "base.h"
#include <algorithm>
#include <stdexcept>
#include <numeric>
#include <clocale>
#include "to_tex.h"

double a, b, q;
int DAYS = 500;
int MODE;
int MODE_2;
int N;
Matrix<int> D;
Matrix<double> T;
double Q;
double l = 0.00000001;

double calculateMedian(vector<int>& v) {
  if (v.empty()) {
    throw invalid_argument();
  }
  std::sort(v.begin(), v.end());
  size_t size = v.size();
  if (size % 2 == 1) {
    return v[size / 2];
  }
  return (v[size / 2 - 1] + v[size / 2]) / 2.0;
```

```
28  }
29
30  double calculateMean(const std::vector<int>& v) {
31    if (v.empty()) {
32      throw invalid_argument();
33    }
34
35    double sum = accumulate(v.begin(), v.end(), 0.0);
36    return sum / v.size();
37  }
38
39  void show_route(vector<int>* nodes) {
40    cout << "---\n";
41    for (int i = 0; i < (nodes->size() - 1); i++) {
42      cout << (*nodes)[i]  << "->" << (*nodes)[i + 1] << ": " << D.
          array[(*nodes)[i]][(*nodes)[i + 1]] << endl;
43    }
44    if (MODE == 1) cout << (*nodes)[(nodes->size() - 1)] << "->" << (*
       nodes)[0] << ": " << D.array[(*nodes)[(nodes->size() - 1)]][(*
       nodes)[0]] << endl;
45  }
46
47  int get_length(vector<int>* nodes) {
48    int length = 0;
49    for (int i = 0; i < (nodes->size() - 1); i++) {
50      length += D.array[(*nodes)[i]][(*nodes)[i + 1]];
51    }
52    if (MODE == 1) length += D.array[(*nodes)[(nodes->size() - 1)]][(*
       nodes)[0]];
53    return length;
54  }
55
56  void count_Q() {
57    for (int i = 0; i < N; i++) {
58      for (int j = 0; j < N; j++) {
59        Q += D.array[i][j];
60      }
61    }
62    if (MODE == 1) Q /= N;
63    else Q /= (N - 1);
64  }
```

```
65
66  void upgrate_pheromon(vector<vector<int>>* routes, int L_best) {
67    //cout << "\nAMOUNT OF THE BEST ROUTES: " << routes->size();
68    for (int i = 0; i < routes->size(); i++) {
69      for (int j = 0; j < N; j++) {
70        T.array[i][j] += Q / L_best;
71      }
72    }
73  }
74
75  int algo(int m) {
76    vector<int> L(N, 0);
77    int L_b = 100000000000;
78    vector<vector<int>> best_routes;
79    int cur;
80    double n;
81    double choose;
82    double sum_prob = 0;
83    Matrix<int> M_h(N);
84    Matrix<double> local_T(N);
85
86    for (int t = 0; t < DAYS; t++) {
87      vector<vector<int>> routes(N, vector<int>(0));
88      for (int k = 0; k < N; k++) {
89        vector<double> P(N, 0);
90        cur = k;
91        while (routes[k].size() < N) {
92          sum_prob = 0;
93          routes[k].push_back(cur);
94
95          for (int g = 0; g < N; g++) {
96            if (find(routes[k].begin(), routes[k].end(), g) != routes[k
                ].end()) P[g] = 0;
97            else {
98              n = 1.0 / D.array[cur][g];// k->g
99              P[g] = pow(n, a) * pow(T.array[cur][g], b);
100             sum_prob += P[g];
101           }
102         }
103         if (sum_prob == 0) break;
104         for (int g = 0; g < N; g++) {
```

```
105             if (find(routes[k].begin(), routes[k].end(), g) != routes[k
                    ].end()) P[g] = 0;
106             else {
107               P[g] /= sum_prob;
108             }
109           }
110           choose = get_random_number(1.0);
111           for (int i = 0; i < N; i++) {
112             choose -= P[i];
113             if (choose < 0) {
114               M_h.array[cur][i] = 1;
115               cur = i;
116               break;
117             }
118           }
119         }
120
121         for (int i = 0; i < N; i++) {
122           for (int j = 0; j < N; j++) {
123             if (M_h.array[i][j] != 0) local_T.array[i][j] = Q /
                    get_length(&routes[k]);
124           }
125         }
126
127         M_h.zero();
128       }
129
130
131       for (int i = 0; i < N; i++) {
132         if (get_length(&routes[i]) < L_b) {
133           L_b = get_length(&routes[i]);
134           best_routes.clear();
135           best_routes.push_back(routes[i]);
136         }
137         else if (get_length(&routes[i]) == L_b) {
138           if (best_routes.front() != routes[i]) best_routes.push_back(
                  routes[i]);
139         }
140       }
141       if (m == 1) {
142         cout << "\n" << L_b << ":\n";
```

```
143         for (int i = 0; i < best_routes.size(); i++)
144         {
145           show_route(&best_routes[i]);
146         }
147       }
148
149       for (int i = 0; i < N; i++) {
150         for (int j = 0; j < N; j++) {
151           T.array[i][j] = T.array[i][j] * (1.0 - q) + local_T.array[i][
                j];
152           if (T.array[i][j] < l) T.array[i][j] = l;
153         }
154       }
155
156       local_T.zero();
157
158       if (MODE_2 == 2) upgrate_pheromon(&best_routes, L_b);
159     }
160     return L_b;
161   }
162
163   void researching() {
164     string fileName = "result.txt";
165     ofstream outFile(fileName);
166
167     if (!outFile.is_open()) {
168       cerr << "ERROR! UNABLE TO OPEN THE FILE.\n";
169     }
170
171     int L_BEST = 1000000000;
172     vector<string> database = {"tests/_5_15.txt", "tests/_4_10.txt", "
          tests/_2_10.txt"};
173     int number_of_file;
174     cout << "ENTER THE NUMBER OF FILE(1, 2, 3): ";
175     cin >> number_of_file;
176     string FILE = database[number_of_file - 1];
177     D = read_matrix(FILE);
178     N = D.size;
179     count_Q();
180     Matrix<double> new_m(N);
181     T = new_m;
```

```
182     int local_best = 100000;
183     cout << "ENTER THE MODE(CYCLE OR ROUTE): ";
184     cin >> MODE;
185     cout << "ENTER THE MODE_2(USUAL OR WUTH ELITE ANTS): ";
186     cin >> MODE_2;
187     vector<int> res;
188     for (double i = 0.200001; i <= 0.9999; i += 0.25) {
189       a = i;
190       for (double j = 0.200001; j <= 0.9999; j += 0.25) {
191         q = j;
192         outFile << "a: " << a << "\nq: " << q << "\n";
193         cout << "a: " << a << "\nq: " << q << "\n";
194         for (int d = 200; d < 1000; d += 300) {
195           cout << "DAYS---------------------------" << d << "\n";
196           outFile << "DAYS---------------------------" << d << "\n";
197           DAYS = d;
198           b = 1.0 - a;
199           for (int t = 0; t < 5; t++) {
200             res.push_back(algo(2));
201             cout << (t + 1) << ") MATRIX: " << FILE << "\nMODE: " <<
                    MODE << "\nMODE_2: " << MODE_2 << "\nRESULT: " << res.
                    back() << "\n";
202             if (local_best > res.back()) local_best = res.back();
203             if (L_BEST > res.back()) L_BEST = res.back();
204           }
205           cout << "BEST:" << local_best << "\n";
206           cout << "AVERAGE: " << calculateMean(res) <<
                    "++++++++++++++++++++++++++\n";
207           cout << "MEDIANA: " << calculateMedian(res) <<
                    "++++++++++++++++++++++++++\n";
208           cout << "\n";
209           outFile << "BEST:" << local_best << "\n";
210           outFile << "AVERAGE:" << calculateMean(res) << "\n";
211           outFile << "MEDIANA:" << calculateMedian(res) << "\n";
212           outFile << "\n";
213           res.clear();
214           local_best = 1000000;
215         }
216         cout << "\n";
217       }
218     }
```

```
219     cout << "\n THE BEST RESULT: " << L_BEST;
220     outFile << L_BEST;
221     outFile.close();
222     create_tex();
223   }
224
225   void prgram(int m) {
226     if (m == 1){
227       D = read_matrix("tests/_2_10.txt");
228       D.print();
229       cout << "ENTER THE FIRST PARAMETR:\n";
230       read_num(&a);
231       b = 1 - a;
232       cout << "a = " << a << "\nb = " << b << "\n";
233       cout << "ENTER THE EVAPORATION COEFFICIENT:\n";
234       read_num(&q);
235       N = D.size;
236       Matrix<double> new_m(N);
237       T = new_m;
238
239       cout << "CHOOSE THE MODE:\n1 - HAMILTONIAN CYCLE\n2 - SHORTEST
             ROUTE\n";
240       while (MODE == 0) {
241         cin >> MODE;
242         if ((MODE != 1) and (MODE != 2)) {
243           cout << "ERROR, TRY AGAIN!\n";
244           MODE = 0;
245         }
246       }
247
248       cout << "CHOOSE THE MODE:\n1- USUAL ALGORITM\n2- ALGORITM WITH
             ELITE ANTS\n";
249       while (MODE_2 == 0) {
250         cin >> MODE_2;
251         if ((MODE_2 != 1) and (MODE_2 != 2)) {
252           cout << "ERROR, TRY AGAIN!\n";
253           MODE_2 = 0;
254         }
255       }
256
257       count_Q();
```

```
258         algo(1);
259       }
260       else {
261         N = D.size;
262         Matrix<double> new_m(N);
263         T = new_m;
264
265         count_Q();
266         algo(2);
267       }
268
269     }
270
271     int main()
272     {
273       int m = 0;
274       cout << "1 - program\n2 - research\n";
275       while (m == 0) {
276         cin >> m;
277         if ((m != 1) and (m != 2)) {
278           cout << "ERROR, TRY AGAIN!\n";
279           m = 0;
280         }
281       }
282       if (m == 1) prgram(1);
283       else researching();
284     }
```

4 Исследовательская часть

Цель исследования — выявление наиболее подходящих параметров для решения задачи коммивояжёра в исследуемых алгоритмах. По результатам параметризации представленных в приложении 4.1 можно выделить несколько случаев наиболее выгодных параметров в соответствии со следующими задачами: наиболее быстрое выявление кратчайшего маршрута и выявление наиболее точного наименьшего маршрута. Реализации с параметрами ($a = 0.45, q = 0.95$), ($a = 0.7, q = 0.45$), ($a = 0.7, q = 0.7$), ($a = 0.7, q = 0.95$), ($a = 0.95, q = 0.45$), находит наилучший маршрут за наименьшее количество циклов, то есть при пяти попытках запуска программы в среднем хотя бы один путь будет наименьшим. Но даже при 800 итерациях, при параметрах ($a = 0.45, q = 0.95$), ($a = 0.7, q = 0.45$), ($a = 0.7, q = 0.7$), ($a = 0.95, q = 0.45$), будет находиться как минимум один не оптимальный маршрут. Реализация с параметрами $a = 0.7, b = 0.3, q = 0.95$ показала наиболее высокую точность. При лучшем маршруте длинной в 275, среднее по пяти попыткам этой реализации составило 277 при 200 итерациях, 276 при 500 итерациях и 275 при 800 итерациях. Следовательно данные параметры наиболее подходящие для выявления наилучшего маршрута с наибольшей точностью, но реализации с такими параметрами необходимо большее количество циклов.

4.1 Вывод

Провели исследование параметризации реализованного алгоритма и нашли два класса параметров наиболее точно соответствующих поставленным задачам.

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы был изучен алгоритм муравьиный алгоритм и его модернизация. Было проведено исследование с целью выявления наиболее подходящих параметров для решения задачи коммивояжёра.

Для достижения поставленной цели были успешно выполнены основные задачи:

1) описаны все нужные понятия для реализации алгоритмов;
2) реализованы алгоритмы;
3) проведено тестирование полученной реализации;
4) проведено исследования с целью выявления наилучших параметров для решения задачи коммивояжёра на определённом кластере данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Иванов И.И. Муравьиные алгоритмы // Журнал "Математические модели". — 2020. — №3. — С. 5–15.(дата обращения: 23.12.2024).

Приложение А

В данном приложении представлены результаты работы реализаций муравьиного алгоритма и его модернизации(алгоритм с элитными муравьями), в зависимости от параметров.

1) Граф из файла `_5_15.txt`(15 вершин) обычный алгоритм:
   - 1.1) $a = 0.200001, q = 0.200001$:
     - — 200 days: $averange = 648.6$, $mediana = 650$, $best = 629$;
     - — 500 days: $averange = 637.2$, $mediana = 641$, $best = 615$;
     - — 800 days: $averange = 611$, $mediana = 616$, $best = 581$;
   - 1.2) $a = 0.200001, q = 0.450001$:
     - — 200 days: $averange = 632.4$, $mediana = 633$, $best = 617$;
     - — 500 days: $averange = 621.2$, $mediana = 621$, $best = 608$;
     - — 800 days: $averange = 610.8$, $mediana = 607$, $best = 598$;
   - 1.3) $a = 0.200001, q = 0.700001$:
     - — 200 days: $averange = 615.8$, $mediana = 614$, $best = 606$;
     - — 500 days: $averange = 620.8$, $mediana = 618$, $best = 614$;
     - — 800 days: $averange = 619.4$, $mediana = 625$, $best = 592$;
   - 1.4) $a = 0.200001, q = 0.950001$:
     - — 200 days: $averange = 651.8$, $mediana = 645$, $best = 640$;
     - — 500 days: $averange = 620$, $mediana = 624$, $best = 603$;
     - — 800 days: $averange = 610.8$, $mediana = 608$, $best = 592$;
   - 1.5) $a = 0.450001, q = 0.200001$:
     - — 200 days: $averange = 630.8$, $mediana = 628$, $best = 616$;
     - — 500 days: $averange = 618.4$, $mediana = 620$, $best = 601$;
     - — 800 days: $averange = 622.2$, $mediana = 619$, $best = 601$;
   - 1.6) $a = 0.450001, q = 0.450001$:
     - — 200 days: $averange = 627$, $mediana = 633$, $best = 606$;
     - — 500 days: $averange = 616.6$, $mediana = 618$, $best = 603$;
     - — 800 days: $averange = 609.2$, $mediana = 608$, $best = 603$;
   - 1.7) $a = 0.450001, q = 0.700001$:
     - — 200 days: $averange = 629.8$, $mediana = 629$, $best = 617$;
     - — 500 days: $averange = 616$, $mediana = 616$, $best = 610$;
     - — 800 days: $averange = 610$, $mediana = 608$, $best = 602$;
   - 1.8) $a = 0.450001, q = 0.950001$:
     - — 200 days: $averange = 623.8$, $mediana = 628$, $best = 613$;
     - — 500 days: $averange = 604.2$, $mediana = 608$, $best = 591$;
     - — 800 days: $averange = 604$, $mediana = 605$, $best = 579$;
   - 1.9) $a = 0.700001, q = 0.200001$:

— 200 days: $averange = 618.4$, $mediana = 619$, $best = 602$;

— 500 days: $averange = 616$, $mediana = 627$, $best = 587$;

— 800 days: $averange = 610.4$, $mediana = 609$, $best = 600$;

1.10) $a = 0.700001, q = 0.450001$:

— 200 days: $averange = 620$, $mediana = 620$, $best = 601$;

— 500 days: $averange = 605.8$, $mediana = 609$, $best = 586$;

— 800 days: $averange = 608.4$, $mediana = 609$, $best = 602$;

1.11) $a = 0.700001, q = 0.700001$:

— 200 days: $averange = 620.8$, $mediana = 621$, $best = 614$;

— 500 days: $averange = 601.6$, $mediana = 606$, $best = 589$;

— 800 days: $averange = 605$, $mediana = 603$, $best = 599$;

1.12) $a = 0.700001, q = 0.950001$:

— 200 days: $averange = 606.8$, $mediana = 609$, $best = 583$;

— 500 days: $averange = 593.4$, $mediana = 597$, $best = 581$;

— 800 days: $averange = 596$, $mediana = 601$, $best = 581$;

1.13) $a = 0.950001, q = 0.200001$:

— 200 days: $averange = 616$, $mediana = 614$, $best = 595$;

— 500 days: $averange = 609.6$, $mediana = 610$, $best = 601$;

— 800 days: $averange = 596.4$, $mediana = 592$, $best = 580$;

1.14) $a = 0.950001, q = 0.450001$:

— 200 days: $averange = 622.8$, $mediana = 621$, $best = 616$;

— 500 days: $averange = 612.2$, $mediana = 615$, $best = 600$;

— 800 days: $averange = 606.2$, $mediana = 605$, $best = 593$;

1.15) $a = 0.950001, q = 0.700001$:

— 200 days: $averange = 614.8$, $mediana = 613$, $best = 599$;

— 500 days: $averange = 612$, $mediana = 609$, $best = 598$;

— 800 days: $averange = 608.2$, $mediana = 604$, $best = 601$;

1.16) $a = 0.950001, q = 0.950001$:

— 200 days: $averange = 623.6$, $mediana = 623$, $best = 614$;

— 500 days: $averange = 604.2$, $mediana = 597$, $best = 593$;

— 800 days: $averange = 593.6$, $mediana = 596$, $best = 578$;

2) Граф из файла `_5_15.txt`(15 вершин) алгоритм с элитными муравьями:

2.1) $a = 0.200001, q = 0.200001$:

— 200 days: $averange = 635.8$, $mediana = 633$, $best = 614$;

— 500 days: $averange = 625$, $mediana = 619$, $best = 605$;

— 800 days: $averange = 623.4$, $mediana = 618$, $best = 613$;

2.2) $a = 0.200001, q = 0.450001$:

— 200 days: $averange = 635.8$, $mediana = 636$, $best = 623$;

— 500 days: $averange = 625.2$, $mediana = 630$, $best = 594$;

— 800 days: $averange = 623.8$, $mediana = 620$, $best = 613$;

2.3) $a = 0.200001, q = 0.700001$:

— 200 days: $averange = 634.8$, $mediana = 644$, $best = 606$;

— 500 days: $averange = 624.2$, $mediana = 629$, $best = 614$;

— 800 days: $averange = 622.2$, $mediana = 623$, $best = 606$;

2.4) $a = 0.200001, q = 0.950001$:

— 200 days: $averange = 631.4$, $mediana = 632$, $best = 608$;

— 500 days: $averange = 625.4$, $mediana = 626$, $best = 615$;

— 800 days: $averange = 622$, $mediana = 628$, $best = 604$;

2.5) $a = 0.450001, q = 0.200001$:

— 200 days: $averange = 625.2$, $mediana = 623$, $best = 619$;

— 500 days: $averange = 616.8$, $mediana = 619$, $best = 597$;

— 800 days: $averange = 616.6$, $mediana = 616$, $best = 610$;

2.6) $a = 0.450001, q = 0.450001$:

— 200 days: $averange = 624.4$, $mediana = 627$, $best = 598$;

— 500 days: $averange = 615$, $mediana = 613$, $best = 612$;

— 800 days: $averange = 611.8$, $mediana = 607$, $best = 598$;

2.7) $a = 0.450001, q = 0.700001$:

— 200 days: $averange = 620$, $mediana = 621$, $best = 617$;

— 500 days: $averange = 609$, $mediana = 610$, $best = 592$;

— 800 days: $averange = 610.4$, $mediana = 612$, $best = 603$;

2.8) $a = 0.450001, q = 0.950001$:

— 200 days: $averange = 624$, $mediana = 622$, $best = 613$;

— 500 days: $averange = 618.6$, $mediana = 620$, $best = 610$;

— 800 days: $averange = 590.2$, $mediana = 594$, $best = 572$;

2.9) $a = 0.700001, q = 0.200001$:

— 200 days: $averange = 612.6$, $mediana = 615$, $best = 584$;

— 500 days: $averange = 618.8$, $mediana = 616$, $best = 610$;

— 800 days: $averange = 607.4$, $mediana = 609$, $best = 593$;

2.10) $a = 0.700001, q = 0.450001$:

— 200 days: $averange = 615.2$, $mediana = 624$, $best = 597$;

— 500 days: $averange = 610.4$, $mediana = 608$, $best = 593$;

— 800 days: $averange = 603.6$, $mediana = 608$, $best = 578$;

2.11) $a = 0.700001, q = 0.700001$:

— 200 days: $averange = 616.4$, $mediana = 619$, $best = 600$;

— 500 days: $averange = 603.2$, $mediana = 608$, $best = 588$;

— 800 days: $averange = 601.4$, $mediana = 600$, $best = 593$;

2.12) $a = 0.700001, q = 0.950001$:

— 200 days: $averange = 601$, $mediana = 603$, $best = 584$;

— 500 days: $averange = 593.8$, $mediana = 600$, $best = 571$;

— 800 days: $averange = 594.4$, $mediana = 597$, $best = 575$;

2.13) $a = 0.950001, q = 0.200001$:

— 200 days: $averange = 613.8$, $mediana = 615$, $best = 601$;

— 500 days: $averange = 604.8$, $mediana = 602$, $best = 593$;

— 800 days: $averange = 603.2$, $mediana = 604$, $best = 592$;

2.14) $a = 0.950001, q = 0.450001$:

— 200 days: $averange = 620.2$, $mediana = 621$, $best = 603$;

— 500 days: $averange = 606$, $mediana = 605$, $best = 593$;

— 800 days: $averange = 600$, $mediana = 600$, $best = 582$;

2.15) $a = 0.950001, q = 0.700001$:

— 200 days: $averange = 612.6$, $mediana = 613$, $best = 604$;

— 500 days: $averange = 611.6$, $mediana = 618$, $best = 591$;

— 800 days: $averange = 601.4$, $mediana = 607$, $best = 586$;

2.16) $a = 0.950001, q = 0.950001$:

— 200 days: $averange = 620$, $mediana = 615$, $best = 613$;

— 500 days: $averange = 603.2$, $mediana = 598$, $best = 590$;

— 800 days: $averange = 603.8$, $mediana = 607$, $best = 593$;

3) Граф из файла `_4_10.txt`(10 вершин) обычный алгоритм(наименьший маршрут длины 305):

3.1) $a = 0.200001, q = 0.200001$:

— 200 days: $averange = 320$, $mediana = 322$, $best = 313$;

— 500 days: $averange = 311.8$, $mediana = 312$, $best = 305$;

— 800 days: $averange = 308.4$, $mediana = 306$, $best = 305$;

3.2) $a = 0.200001, q = 0.450001$:

— 200 days: $averange = 318$, $mediana = 318$, $best = 315$;

— 500 days: $averange = 308.6$, $mediana = 308$, $best = 305$;

— 800 days: $averange = 310$, $mediana = 312$, $best = 305$;

3.3) $a = 0.200001, q = 0.700001$:

— 200 days: $averange = 318.4$, $mediana = 321$, $best = 305$;

— 500 days: $averange = 310.6$, $mediana = 308$, $best = 305$;

— 800 days: $averange = 308.8$, $mediana = 308$, $best = 305$;

3.4) $a = 0.200001, q = 0.950001$:

— 200 days: $averange = 313.2$, $mediana = 313$, $best = 306$;

— 500 days: $averange = 311.6$, $mediana = 313$, $best = 306$;

— 800 days: $averange = 314.4$, $mediana = 313$, $best = 308$;

3.5) $a = 0.450001, q = 0.200001$:

— 200 days: $averange = 309$, $mediana = 308$, $best = 306$;

— 500 days: $averange = 307.2$, $mediana = 306$, $best = 305$;

— 800 days: $averange = 306$, $mediana = 306$, $best = 305$;

3.6) $a = 0.450001, q = 0.450001$:

— 200 days: $averange = 310.6$, $mediana = 313$, $best = 306$;

— 500 days: $averange = 309.4$, $mediana = 308$, $best = 306$;

— 800 days: $averange = 305.2$, $mediana = 305$, $best = 305$;

3.7) $a = 0.450001, q = 0.700001$:

— 200 days: $averange = 307.2$, $mediana = 306$, $best = 305$;

— 500 days: $averange = 306.8$, $mediana = 306$, $best = 305$;

— 800 days: $averange = 308.6$, $mediana = 308$, $best = 305$;

3.8) $a = 0.450001, q = 0.950001$:

— 200 days: $averange = 310$, $mediana = 312$, $best = 305$;

— 500 days: $averange = 309$, $mediana = 308$, $best = 305$;

— 800 days: $averange = 305.4$, $mediana = 305$, $best = 305$;

3.9) $a = 0.700001, q = 0.200001$:

— 200 days: $averange = 312.2$, $mediana = 313$, $best = 308$;

— 500 days: $averange = 310.2$, $mediana = 312$, $best = 305$;

— 800 days: $averange = 305.8$, $mediana = 305$, $best = 305$;

3.10) $a = 0.700001, q = 0.450001$:

— 200 days: $averange = 308$, $mediana = 306$, $best = 305$;

— 500 days: $averange = 307.6$, $mediana = 306$, $best = 305$;

— 800 days: $averange = 305.2$, $mediana = 305$, $best = 305$;

3.11) $a = 0.700001, q = 0.700001$:

— 200 days: $averange = 310.6$, $mediana = 313$, $best = 305$;

— 500 days: $averange = 305.8$, $mediana = 305$, $best = 305$;

— 800 days: $averange = 305.6$, $mediana = 306$, $best = 305$;

3.12) $a = 0.700001, q = 0.950001$:

— 200 days: $averange = 307.8$, $mediana = 305$, $best = 305$;

— 500 days: $averange = 305.2$, $mediana = 305$, $best = 305$;

— 800 days: $averange = 305$, $mediana = 305$, $best = 305$;

3.13) $a = 0.950001, q = 0.200001$:

— 200 days: $averange = 308.6$, $mediana = 308$, $best = 305$;

— 500 days: $averange = 307.2$, $mediana = 305$, $best = 305$;

— 800 days: $averange = 306.4$, $mediana = 305$, $best = 305$;

3.14) $a = 0.950001, q = 0.450001$:

— 200 days: $averange = 308.8$, $mediana = 308$, $best = 305$;

— 500 days: $averange = 307$, $mediana = 306$, $best = 305$;

— 800 days: $averange = 305$, $mediana = 305$, $best = 305$;

3.15) $a = 0.950001, q = 0.700001$:

— 200 days: $averange = 311.2$, $mediana = 313$, $best = 306$;

— 500 days: $averange = 305.8$, $mediana = 305$, $best = 305$;

— 800 days: $averange = 307.2$, $mediana = 306$, $best = 305$;

3.16) $a = 0.950001, q = 0.950001$:

— 200 days: $averange = 311.4$, $mediana = 312$, $best = 308$;

— 500 days: $averange = 307$, $mediana = 306$, $best = 305$;

— 800 days: $averange = 305.6$, $mediana = 305$, $best = 305$;

4) Граф из файла `_4_10.txt`(10 вершин) алгоритм с элитными муравьями:

4.1) $a = 0.200001, q = 0.200001$:

— 200 days: $averange = 315.4$, $mediana = 318$, $best = 306$;

— 500 days: $averange = 314.4$, $mediana = 316$, $best = 306$;

— 800 days: $averange = 311.2$, $mediana = 312$, $best = 306$;

4.2) $a = 0.200001, q = 0.450001$:

— 200 days: $averange = 313.8$, $mediana = 316$, $best = 306$;

— 500 days: $averange = 309$, $mediana = 308$, $best = 305$;

— 800 days: $averange = 306.6$, $mediana = 305$, $best = 305$;

4.3) $a = 0.200001, q = 0.700001$:

— 200 days: $averange = 314.6$, $mediana = 313$, $best = 308$;

— 500 days: $averange = 307.8$, $mediana = 308$, $best = 305$;

— 800 days: $averange = 312$, $mediana = 313$, $best = 306$;

4.4) $a = 0.200001, q = 0.950001$:

— 200 days: $averange = 319.8$, $mediana = 320$, $best = 305$;

— 500 days: $averange = 311.8$, $mediana = 313$, $best = 305$;

— 800 days: $averange = 314.4$, $mediana = 312$, $best = 312$;

4.5) $a = 0.450001, q = 0.200001$:

— 200 days: $averange = 315.6$, $mediana = 318$, $best = 305$;

— 500 days: $averange = 307.4$, $mediana = 306$, $best = 305$;

— 800 days: $averange = 306.8$, $mediana = 305$, $best = 305$;

4.6) $a = 0.450001, q = 0.450001$:

— 200 days: $averange = 310$, $mediana = 308$, $best = 305$;

— 500 days: $averange = 306.6$, $mediana = 305$, $best = 305$;

— 800 days: $averange = 309$, $mediana = 308$, $best = 305$;

4.7) $a = 0.450001, q = 0.700001$:

— 200 days: $averange = 309.4$, $mediana = 312$, $best = 305$;

— 500 days: $averange = 306.2$, $mediana = 305$, $best = 305$;

— 800 days: $averange = 307.6$, $mediana = 306$, $best = 305$;

4.8) $a = 0.450001, q = 0.950001$:

— 200 days: $averange = 309.4$, $mediana = 308$, $best = 305$;

— 500 days: $averange = 308.4$, $mediana = 308$, $best = 305$;

— 800 days: $averange = 305.8$, $mediana = 306$, $best = 305$;

4.9) $a = 0.700001, q = 0.200001$:

— 200 days: $averange = 313.4$, $mediana = 312$, $best = 306$;

— 500 days: $averange = 307.8$, $mediana = 308$, $best = 305$;

— 800 days: $averange = 308.6$, $mediana = 308$, $best = 305$;

4.10) $a = 0.700001, q = 0.450001$:

— 200 days: $averange = 310.4$, $mediana = 312$, $best = 305$;

— 500 days: $averange = 310.2$, $mediana = 312$, $best = 305$;

— 800 days: $averange = 305.6$, $mediana = 305$, $best = 305$;

4.11) $a = 0.700001, q = 0.700001$:

— 200 days: $averange = 309.6$, $mediana = 308$, $best = 305$;

— 500 days: $averange = 306.4$, $mediana = 305$, $best = 305$;

— 800 days: $averange = 306.8$, $mediana = 305$, $best = 305$;

4.12) $a = 0.700001, q = 0.950001$:

— 200 days: $averange = 306.6$, $mediana = 306$, $best = 305$;

— 500 days: $averange = 306.6$, $mediana = 305$, $best = 305$;

— 800 days: $averange = 305.2$, $mediana = 305$, $best = 305$;

4.13) $a = 0.950001, q = 0.200001$:

— 200 days: $averange = 309.2$, $mediana = 306$, $best = 306$;

— 500 days: $averange = 306.6$, $mediana = 305$, $best = 305$;

— 800 days: $averange = 306.2$, $mediana = 305$, $best = 305$;

4.14) $a = 0.950001, q = 0.450001$:

— 200 days: $averange = 309.2$, $mediana = 308$, $best = 305$;

— 500 days: $averange = 309.2$, $mediana = 308$, $best = 305$;

— 800 days: $averange = 308.2$, $mediana = 306$, $best = 305$;

4.15) $a = 0.950001, q = 0.700001$:

— 200 days: $averange = 306.4$, $mediana = 306$, $best = 305$;

— 500 days: $averange = 307.2$, $mediana = 306$, $best = 305$;

— 800 days: $averange = 305.2$, $mediana = 305$, $best = 305$;

4.16) $a = 0.950001, q = 0.950001$:

— 200 days: $averange = 309.4$, $mediana = 308$, $best = 305$;

— 500 days: $averange = 305.8$, $mediana = 305$, $best = 305$;

— 800 days: $averange = 307.6$, $mediana = 306$, $best = 305$;

5) Граф из файла `_2_10.txt`(10 вершин) обычный алгоритм(длина кратчайшего маршрута равна 275):

5.1) $a = 0.200001, q = 0.200001$:

— 200 days: $averange = 284$, $mediana = 280$, $best = 275$;

— 500 days: $averange = 278$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 280$, $mediana = 280$, $best = 280$;

5.2) $a = 0.200001, q = 0.450001$:

— 200 days: $averange = 285$, $mediana = 280$, $best = 280$;

— 500 days: $averange = 283$, $mediana = 285$, $best = 275$;

— 800 days: $averange = 277$, $mediana = 275$, $best = 275$;

5.3) $a = 0.200001, q = 0.700001$:

— 200 days: $averange = 285$, $mediana = 285$, $best = 280$;

— 500 days: $averange = 281$, $mediana = 280$, $best = 275$;

— 800 days: $averange = 277$, $mediana = 275$, $best = 275$;

5.4) $a = 0.200001, q = 0.950001$:

— 200 days: $averange = 292$, $mediana = 295$, $best = 280$;

— 500 days: $averange = 282$, $mediana = 280$, $best = 275$;

— 800 days: $averange = 283$, $mediana = 285$, $best = 275$;

5.5) $a = 0.450001, q = 0.200001$:

— 200 days: $averange = 280$, $mediana = 280$, $best = 275$;

— 500 days: $averange = 281$, $mediana = 280$, $best = 275$;

— 800 days: $averange = 279$, $mediana = 275$, $best = 275$;

5.6) $a = 0.450001, q = 0.450001$:

— 200 days: $averange = 278$, $mediana = 280$, $best = 275$;

— 500 days: $averange = 278$, $mediana = 280$, $best = 275$;

— 800 days: $averange = 279$, $mediana = 280$, $best = 275$;

5.7) $a = 0.450001, q = 0.700001$:

— 200 days: $averange = 284$, $mediana = 285$, $best = 275$;

— 500 days: $averange = 278$, $mediana = 280$, $best = 275$;

— 800 days: $averange = 280$, $mediana = 280$, $best = 280$;

5.8) $a = 0.450001, q = 0.950001$:

— 200 days: $averange = 283$, $mediana = 285$, $best = 275$;

— 500 days: $averange = 279$, $mediana = 280$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;

5.9) $a = 0.700001, q = 0.200001$:

— 200 days: $averange = 281$, $mediana = 280$, $best = 275$;

— 500 days: $averange = 275$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 275$, $mediana = 275$, $best = 275$;

5.10) $a = 0.700001, q = 0.450001$:

— 200 days: $averange = 281$, $mediana = 285$, $best = 275$;

— 500 days: $averange = 277$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;

5.11) $a = 0.700001, q = 0.700001$:

— 200 days: $averange = 279$, $mediana = 275$, $best = 275$;

— 500 days: $averange = 276$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;

5.12) $a = 0.700001, q = 0.950001$:

— 200 days: $averange = 277$, $mediana = 275$, $best = 275$;

— 500 days: $averange = 276$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 275$, $mediana = 275$, $best = 275$;

5.13) $a = 0.950001, q = 0.200001$:

— 200 days: $averange = 279$, $mediana = 275$, $best = 275$;

— 500 days: $averange = 277$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;

5.14) $a = 0.950001, q = 0.450001$:

— 200 days: $averange = 277$, $mediana = 275$, $best = 275$;

— 500 days: $averange = 277$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 275$, $mediana = 275$, $best = 275$;

5.15) $a = 0.950001, q = 0.700001$:

— 200 days: $averange = 279$, $mediana = 280$, $best = 275$;

— 500 days: $averange = 276$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;

5.16) $a = 0.950001, q = 0.950001$:

— 200 days: $averange = 278$, $mediana = 280$, $best = 275$;

— 500 days: $averange = 276$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 275$, $mediana = 275$, $best = 275$;

6) `_2_10.txt`(10 вершин) алгоритм с элитными муравьями(длина кратчайшего маршрута равна 275):

6.1) $a = 0.200001, q = 0.200001$:

— 200 days: $averange = 285$, $mediana = 285$, $best = 280$;

— 500 days: $averange = 284$, $mediana = 285$, $best = 280$;

— 800 days: $averange = 279$, $mediana = 275$, $best = 275$;

6.2) $a = 0.200001, q = 0.450001$:

— 200 days: $averange = 288$, $mediana = 285$, $best = 280$;

— 500 days: $averange = 282$, $mediana = 280$, $best = 280$;

— 800 days: $averange = 280$, $mediana = 280$, $best = 275$;

6.3) $a = 0.200001, q = 0.700001$:

— 200 days: $averange = 283$, $mediana = 280$, $best = 275$;

— 500 days: $averange = 280$, $mediana = 280$, $best = 275$;

— 800 days: $averange = 279$, $mediana = 275$, $best = 275$;

6.4) $a = 0.200001, q = 0.950001$:

— 200 days: $averange = 292$, $mediana = 295$, $best = 285$;

— 500 days: $averange = 277$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;

6.5) $a = 0.450001, q = 0.200001$:

— 200 days: $averange = 288$, $mediana = 290$, $best = 280$;

— 500 days: $averange = 281$, $mediana = 280$, $best = 280$;

— 800 days: $averange = 279$, $mediana = 280$, $best = 275$;

6.6) $a = 0.450001, q = 0.450001$:

— 200 days: $averange = 279$, $mediana = 280$, $best = 275$;

— 500 days: $averange = 277$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 278$, $mediana = 280$, $best = 275$;

6.7) $a = 0.450001, q = 0.700001$:

— 200 days: $averange = 280$, $mediana = 280$, $best = 275$;

— 500 days: $averange = 277$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 277$, $mediana = 275$, $best = 275$;

6.8) $a = 0.450001, q = 0.950001$:

— 200 days: $averange = 276$, $mediana = 275$, $best = 275$;

— 500 days: $averange = 277$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 275$, $mediana = 275$, $best = 275$;

6.9) $a = 0.700001, q = 0.200001$:

— 200 days: $averange = 280$, $mediana = 280$, $best = 275$;

— 500 days: $averange = 280$, $mediana = 280$, $best = 275$;

— 800 days: $averange = 279$, $mediana = 280$, $best = 275$;

6.10) $a = 0.700001, q = 0.450001$:

— 200 days: $averange = 280$, $mediana = 275$, $best = 275$;

— 500 days: $averange = 279$, $mediana = 280$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;

6.11) $a = 0.700001, q = 0.700001$:

— 200 days: $averange = 278$, $mediana = 280$, $best = 275$;

— 500 days: $averange = 278$, $mediana = 280$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;

6.12) $a = 0.700001, q = 0.950001$:

— 200 days: $averange = 277$, $mediana = 275$, $best = 275$;

— 500 days: $averange = 275$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;

6.13) $a = 0.950001, q = 0.200001$:

— 200 days: $averange = 282$, $mediana = 285$, $best = 275$;

— 500 days: $averange = 277$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;

6.14) $a = 0.950001, q = 0.450001$:

— 200 days: $averange = 278$, $mediana = 275$, $best = 275$;

— 500 days: $averange = 276$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;

6.15) $a = 0.950001, q = 0.700001$:

— 200 days: $averange = 279$, $mediana = 280$, $best = 275$;

— 500 days: $averange = 275$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;

6.16) $a = 0.950001, q = 0.950001$:

— 200 days: $averange = 281$, $mediana = 280$, $best = 280$;

— 500 days: $averange = 277$, $mediana = 275$, $best = 275$;

— 800 days: $averange = 276$, $mediana = 275$, $best = 275$;