Государственное образовательное учреждение высшего
профессионального образования
“Московский государственный технический университет имени
Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа № 6

Муравьиный алгоритм

Гибадулин О.Н.
Студент группы ИУ7-52

2019 г.

Содержание

Введение

Одной из самых известных задач комбинаторной оптимизации является задача коммивояжера. В классической постановке коммивояжеру нужно совершить тур, посетив каждый город ровно по одному разу и завершив путешествие в том же городе, из которого он выехал. Существует несколько методов решения этой задачи, одним из которых является муравьиный алгоритм. Муравьиный алгоритм — один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах.

В данной работе требуется рассмотреть муравьиный алгоритм для решения задачи коммивояжёра.

Цель работы: изучение муравьиного алгоритма и определение его оптимальных параметров.

Задачи работы:

1. разработка и реализация алгоритма;
2. исследование работы алгоритма при различных параметрах;
3. описание и обоснование полученных результатов.

1 Аналитический раздел

В данном разделе будет описан муравьиный алгоритм на примере решения задачи коммивояжера.

1.1 Задача коммивояжера

Задача коммивояжера формулируется как задача поиска минимального по стоимости замкнутого маршрута по всем вершинам без повторений на полном взвешенном графе с n вершинами. Вершины графа являются городами, которые должен посетить коммивояжер, а веса ребер отражают расстояния или стоимости проезда. Эта задача является NP-трудной, и точный переборный алгоритм ее решения имеет факториальную сложность.[1]

1.2 Муравьиный алгоритм

Муравьиные алгоритмы представляют собой вероятностную жадную эвристику, где вероятности устанавливаются, исходя из информации о качестве решения, полученной из предыдущих решений. Идея муравьиного алгоритма - моделирование поведения муравьёв, связанного с их способностью быстро находить кратчайший путь от муравейника к источнику пищи и адаптироваться к изменяющимся условиям, находя новый кратчайший путь.

Моделирование поведения муравьёв связано с распределением феромона на тропе – ребре графа в задаче коммивояжёра. При этом вероятность включения ребра в маршрут отдельного муравья пропорциональна количеству феромона на этом ребре, а количество откладываемого феромона пропорционально длине маршрута. Чем короче маршрут, тем больше феромона будет отложено на его рёбрах, следовательно, большее количество муравьёв будет включать его в синтез собственных маршрутов. Моделирование такого подхода, использующего только положительную обратную связь, приводит к преждевременной сходимости – большинство муравьёв двигается по локально оптимальному маршруту. Избежать этого можно, моделируя отрицательную обратную связь в виде испарения феромона.

С учётом особенностей задачи коммивояжёра, мы можем описать локальные правила поведения муравьёв при выборе пути.

1. Муравьи обладают «памятью». Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещённых городов. Обозначим через $J_{i,k}$ список городов, которые необходимо посетить муравью k, находящемуся в городе i.

2. Муравьи обладают «зрением», которое определяет степень желания посетить город j, если муравей находится в городе i. Будем считать, что видимость обратно пропорциональна расстоянию между городами.

3. Муравьи обладают «обонянием», с помощью которого 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 $\tau_{ij}(t)$.

4. На основании предыдущих утверждений мы можем сформулировать вероятностно-пропорциональное правило, определяющее вероятность перехода k-ого муравья из города i в город j:

$$P_{ij,k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{l \in J_(i,k)} [\tau_{il}(t)]^{\alpha} * [\eta_{il}]^{\beta}}, j \in J_(i,k) \\ 0, j \notin J_(i,k) \end{cases},$$

где $\tau_{ij}(t)$ – уровень феромона, η_{ij} – эвристическое расстояние, а α и β – константные параметры.

Выбор города является вероятностным, в общую зону всех городов бросается случайное число, которое и определяет выбор муравья. При $\alpha = 0$ алгоритм вырождается до жадного алгоритма, по которому на каждом шаге будет выбираться ближайший город.

5. При прохождении ребра муравей оставляет на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть есть маршрут, пройденный муравьём k к моменту времени t, T – длина этого маршрута, $L_k(t)$ - цена текущего решения для k-ого муравья а Q – параметр, имеющий значение порядка цены оптимального решения.

Тогда откладываемое количество феромона:

$$\Delta\tau_{ij,k}(t) = \begin{cases} \frac{Q}{L_k(t)}, (i,j) \in T_k(t) \\ 0, (i,j) \notin T_k(t) \end{cases},$$

и испаряемое количество феромона:

$$\tau_{ij}(t+1) = (1-p) * \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij,k}(t),$$

где m – количество муравьёв в колонии.[2]

1.3 Вывод

В данном разделе был описан муравьиный алгоритм на примере решения задачи коммивояжера.

2 Конструкторский раздел

В данном разделе в соответствии с описанием алгоритмов, приведенными в аналитической части работы, будет рассмотрена схема муравьиного алгоритма.

2.1 Формализация процесса

В данном пункте представлена idef0-диаграмма для описания функциональной модели процесса работы муравьиного алгоритма (рис. 2.1.1, рис. 2.1.2, рис. 2.1.3).

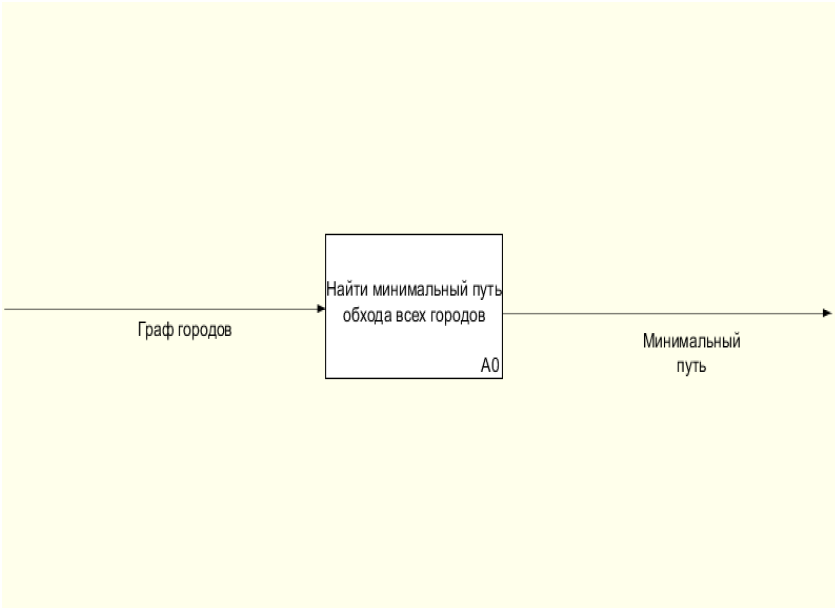

Рисунок 2.1.1. – функциональная схема верхнего уровня процесса работы муравьиного алгоритма

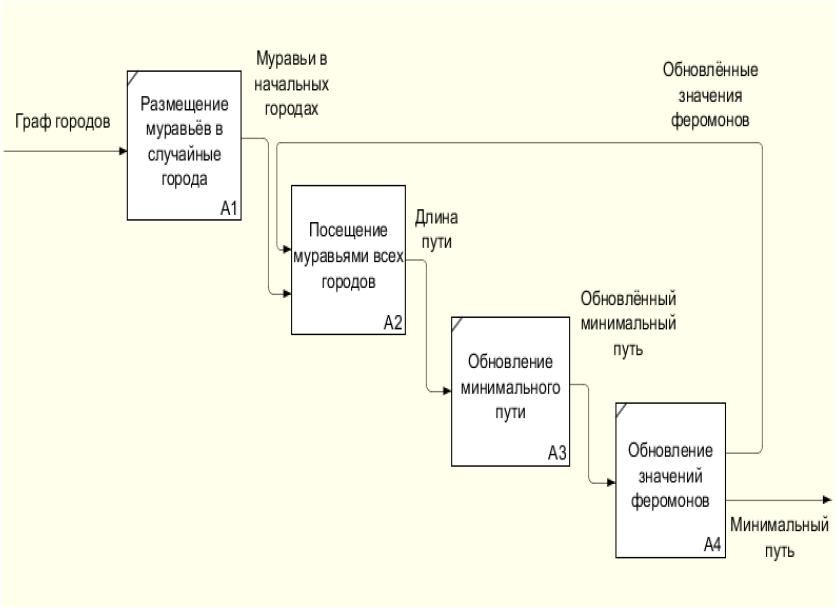

Рисунок 2.1.2. – функциональная схема второго уровня процесса работы муравьиного алгоритма

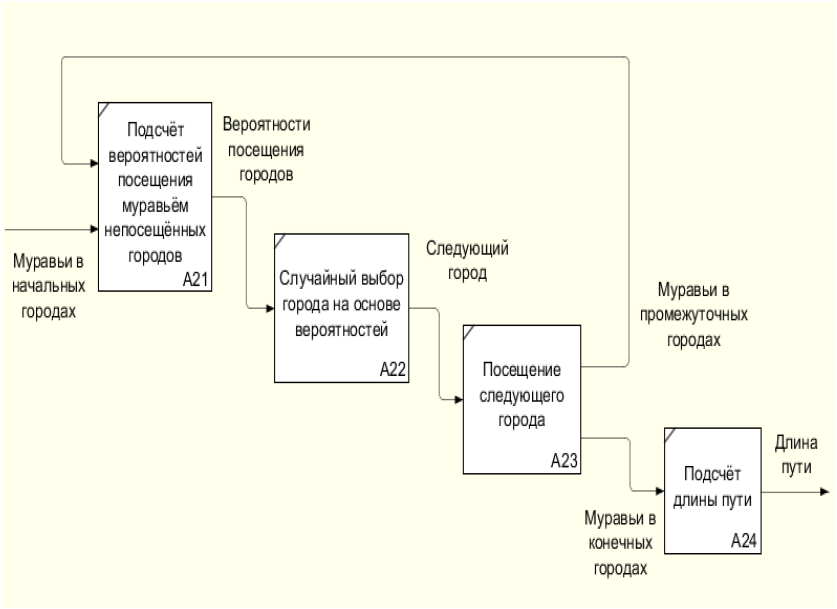

Рисунок 2.1.2. – функциональная схема третьего уровня процесса работы муравьиного алгоритма

2.2 Разработка Алгоритма

В данном пункте представлена схема муравьиного алгоритма.

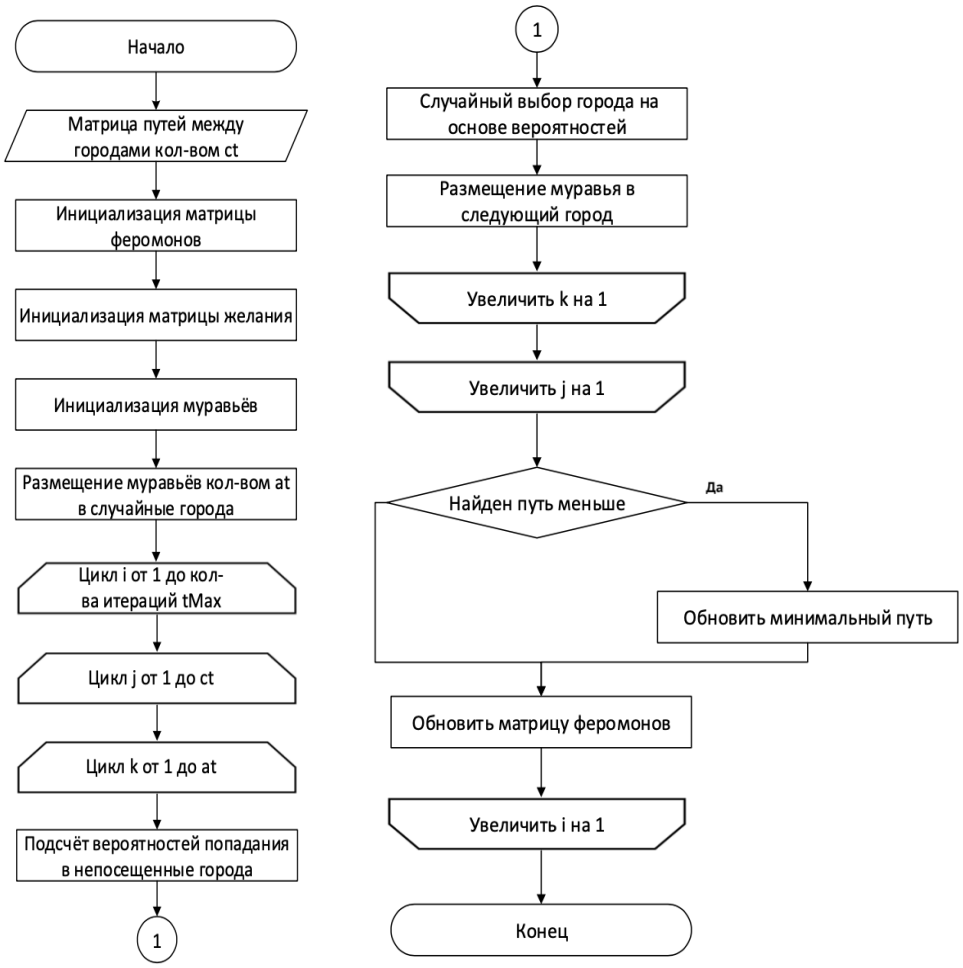

Рисунок 2.2.1. – схема муравьиного алгоритма

2.3 Вывод

В данном разделе была рассмотрена схема муравьиного алгоритма.

3 Технологический раздел

В данном разделе будут рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также представлен листинг кода программы.

3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать муравьиный алгоритм для решения задачи коммивояжера. Пользователь должен иметь возможность произвести вычисления для графов, количество вершин которых он вводит, а также иметь возможность сравнить результаты работы алгоритма при различных параметрах.

3.2 Средства реализации

Для реализации поставленной задачи был использован язык программирования C++[3]. Проет был выполнен в среде XCode[4]. Для измерения процессорного времени была использована ассемблерная инструкция rdtsc[5].

3.3 Листинг кода

На основе схем алгоритмов, представленных в конструкторском разделе, в соответствии с указанными требованиями к реализации с использованием средств языка C++ было разработано программное обеспечение, содержащее реализацию выбранного алгоритма. В данном пункте приведён листиннг этой реализации.

Листинг 1. Код реализации муравьиного алгоритма

```
class Ant {
public:
size_t pathLength;
vector <bool> visited;
vector <size_t> path;

Ant(const size_t graphSize) : pathLength(0) {
for (size_t i = 0; i < graphSize; ++i) {
path.push_back(0);
visited.push_back(false);
}
}

void visitCity(const size_t city, const size_t
    curPathLength, const size_t curPathDistance) {
pathLength += curPathDistance;
path[curPathLength] = city;
```

```
visited[city] = true;
}

void clearVisits() {
for (size_t i = 0; i < visited.size(); ++i) {
visited[i] = false;
}
pathLength = 0;
}

void makeDefaultPath() {
pathLength = 0;
visitCity(path[path.size() - 1], 0, 0);
}

bool isVisited(const size_t city) const {
return visited[city];
}
};


class ACO {
private:
const vector <vector <int>> distGraph;
const size_t citiesCount;

vector <vector <double>> pherGraph;
vector <vector <double>> desireGraph;

vector <Ant> ants;
size_t antsCount;

vector <double> pathsProbabilities;

double alpha = 0.5;
double rho = 0.5;
size_t tMax = 100;
double beta = 1 - alpha;
double Q = 5;
double antsFactor = 1;
double initialPherVal = 1;

public:
size_t minLength = 0;
vector <size_t> minPath;

ACO(const vector<vector <int>>& graph) :
    distGraph(graph), citiesCount(graph.size()) {
// init pherGraph
for (size_t i = 0; i < citiesCount; ++i) {
```

```
vector<double> line;
for (size_t j = 0; j < citiesCount; ++j) {
line.push_back(initialPherVal);
}
pherGraph.push_back(line);
}

// init desireGraph
for (size_t i = 0; i < citiesCount; ++i) {
vector<double> line;
for (size_t j = 0; j < citiesCount; ++j) {
if (distGraph[i][j] == 0) {
line.push_back(0);
}
else {
line.push_back(1.0 / distGraph[i][j]);
}
}
desireGraph.push_back(line);
}

// init antsCount
antsCount = citiesCount * antsFactor;
for (size_t i = 0; i < antsCount; ++i) {
Ant ant(citiesCount);
ants.push_back(ant);
}

// init pathsProbabilities
for (size_t i = 0; i < citiesCount; ++i) {
pathsProbabilities.push_back(0);
}
}

void execute() {
makeDefaultState();
initPherGraph();
initAnts();

for (size_t i = 0; i < tMax; ++i) {
paveAntsPaths();
updateMinPath();
updatePheromones();
makeDefaultAnts();
}
}

void changeParams(double alpha, double rho,
    size_t tMax) {
this->alpha = alpha;
```

```
this->beta = 1 - alpha;
this->rho = rho;
this->tMax = tMax;
}

private:
void makeDefaultState() {
minLength = 0;
minPath.clear();
}

void initAnts() {
for (size_t i = 0; i < antsCount; ++i) {
ants[i].clearVisits();
ants[i].visitCity(rand() % citiesCount, 0, 0);
}
}

void initPherGraph() {
for (size_t i = 0; i < citiesCount; ++i) {
for (size_t j = 0; j < citiesCount; ++j) {
pherGraph[i][j] = initialPherVal;
}
}
}

void paveAntsPaths() {
for (size_t i = 0; i < citiesCount - 1; ++i) {
for (size_t j = 0; j < antsCount; ++j) {
const size_t curCity = ants[j].path[i];
const size_t nextCity = getNextCity(ants[j],
    curCity);
const int distance = distGraph[curCity][nextCity
    ];

ants[j].visitCity(nextCity, i + 1, distance);
}
}

for (size_t j = 0; j < antsCount; ++j) {
const int distToInitialCity = distGraph[ants[j].
    path[ants[j].path.size() - 1]][ants[j].path
    [0]];
ants[j].pathLength += distToInitialCity;
}
}

size_t getNextCity(const Ant& ant, const size_t
    curCity) {
double sumP = 0;
```

```
for (int i = 0; i < citiesCount; ++i) {
sumP += pow(pherGraph[curCity][i], alpha) * pow(
    desireGraph[curCity][i], beta);
}

for (int i = 0; i < citiesCount; ++i) {
if (i == curCity || ant.isVisited(i)) {
pathsProbabilities[i] = 0;
}
else {
pathsProbabilities[i] = pow(pherGraph[curCity][i
    ], alpha) * pow(desireGraph[curCity][i], beta)
     / sumP;
}
}

size_t nextCity = selectNextCity();

return nextCity;
}

void updateMinPath() {
for (size_t i = 0; i < antsCount; ++i) {
const size_t curLength = ants[i].pathLength;
if (curLength < minLength || minLength == 0) {
minLength = curLength;
minPath = ants[i].path;
}
}
}

void updatePheromones() {
for (int i = 0; i < citiesCount; ++i) {
for (int j = 0; j < citiesCount; ++j) {
pherGraph[i][j] *= (1 - rho);
}
}

for (size_t i = 0; i < antsCount; ++i) {
double deltaTau = Q / ants[i].pathLength;
for (int j = 0; j < citiesCount - 1; ++j) {
pherGraph[ants[i].path[j]][ants[i].path[j + 1]]
    += deltaTau;
}
pherGraph[ants[i].path[citiesCount - 1]][ants[i].
    path[0]] += deltaTau;
}
}
```

```
        void makeDefaultAnts() {
        for (size_t i = 0; i < antsCount; ++i) {
        ants[i].clearVisits();
        ants[i].makeDefaultPath();
        }
        }

        size_t selectNextCity() {
        double sumProbabilities = getSumProbabilities();

        double randNum = ((double) rand() / (RAND_MAX)) *
            sumProbabilities;
        double total = 0;
        size_t city = 0;

        for (size_t i = 0; i < citiesCount; ++i) {
        total += pathsProbabilities[i];
        if (total >= randNum) {
        city = i;
        break;
        }
        }

        return city;
        }

        double getSumProbabilities() {
        double sumProbabilities = 0;

        for (size_t i = 0; i < citiesCount; ++i) {
        sumProbabilities += pathsProbabilities[i];
        }

        return sumProbabilities;
        }
        };
```

3.4 Вывод

В данном разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки, а также был представлен листинг реализации муравьинного алгоритма.

4 Экспериментальный раздел

В данном разделе будет проведен сравнительный анализ работы реализованного муравьинного алгоритма при различных параметрах.

4.1 Сравнительное исследование

Для сравнения работы муравьиного алгоритма при различных параметрах (табл. 4.1.1) замеры выполнялись на графе из 15 узлов. Параметры α и η варьируются от 0 до 1 с шагом 0.3, а количество итераций tMax - от 100 до 400 с шагом 100.

Таблица 4.1.1. Сравнение работы муравьиного алгоритма при различных параметрах

α	η	tMax	Мин. путь	α	η	tMax	Мин. путь
0	0	100	26	0.6	0.6	100	26
0	0	200	22	0.6	0.6	200	26
0	0	300	23	0.6	0.6	300	23
0	0.3	100	25	0.6	0.9	100	23
0	0.3	200	22	0.6	0.9	200	24
0	0.3	300	23	0.6	0.9	300	26
0	0.6	100	24	0.9	0	100	36
0	0.6	200	24	0.9	0	200	29
0	0.6	300	21	0.9	0	300	31
0	0.9	100	29	0.9	0.3	100	32
0	0.9	200	23	0.9	0.3	200	30
0	0.9	300	28	0.9	0.3	300	28
0.3	0	100	23	0.9	0.6	100	33
0.3	0	200	27	0.9	0.6	200	26
0.3	0	300	22	0.9	0.6	300	30
0.3	0.3	100	23	0.9	0.9	100	37
0.3	0.3	200	23	0.9	0.9	200	33
0.3	0.3	300	26	0.9	0.9	300	24
0.3	0.6	100	26	1	0	100	45
0.3	0.6	200	24	1	0	200	38
0.3	0.6	300	20	1	0	300	41
0.3	0.9	100	22	1	0.3	100	44
0.3	0.9	200	21	1	0.3	200	31
0.3	0.9	300	21	1	0.3	300	37
0.6	0	100	24	1	0.6	100	38
0.6	0	200	30	1	0.6	200	40
0.6	0	300	22	1	0.6	300	35
0.6	0.3	100	29	1	0.9	100	36
0.6	0.3	200	24	1	0.9	200	41
0.6	0.3	300	26	1	0.9	300	37

Из данной таблицы можно увидеть, что для данного набора параметров при $\alpha = 1$, $\eta = 0$ и tMax $= 100$ муравьиный алгоритм выдает наихудший результат. При параметрах $\alpha = 0.3$, $\eta = 0.6$ и tMax $= 300$ муравьинный алгоритм наиболее приближен к результату, полученному полным перебором. При правильном подборе параметров муравьиный алгоритм выдает результат, близкий к наилучшему, при этом работая намного быстрее полного перебора (на 99.6% быстрее на графе из 10 узлов).

4.2 Вывод

В данном разделе был проведен сравнительный анализ работы реализованного муравьинного алгоритма при различных параметрах, из которого можно сделать вывод, что при правильном подборе параметров муравьиный алгоритм находит оптимальный ответ за приемлимое время, намного отличающееся (на 99.6% быстрее на графе из 10 узлов) от времени нахождения пути полным перебором.

Заключение

В ходе выполнения данной лабораторной работы был изучен и реализован муравьиный алгоритм. Алгоритм был разработан и реализацован для решения задачи коммивояжера, было проведено исследование работы алгоритма при различных параметрах, а также дано описание и обоснование полученных результатов.

В аналитическом разделе было дано описание стандартного алгоритма и алгоритма Винограда. В конструкторском разделе был формализован и описан процесс вычисления пороизвдения двух матриц, разработаны алгоритмы. В технологическом разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также представлен листинг кода программы. В экспериментальном разеде было проведено исследование временных затрат.

Список литературы

[1] Шутова Ю.О., Мартынова Ю.А. ИССЛЕДОВАНИЕ ВЛИЯНИЯ РЕГУЛИРУЕМЫХ ПАРАМЕТРОВ МУРАВЬИНОГО АЛГОРИТМА НА СХОДИМОСТЬ. Томский политехнический университет, 634050, Россия, г. Томск, пр. Ленина, 30, 2014. С. 281-282.

[2] Чураков Михаил, Якушев Андрей Муравьиные алгоритмы. 2006. С. 9-11.

[3] ISO/IEC JTC1 SC22 WG21 N 3690 «Programming Languages — C++» [Электронный ресурс]. – Режим доступа: https://devdocs.io/cpp/, свободный. (Дата обращения: 29.09.2019 г.)

[4] Apple «Apple Developer Documentation» [Электронный ресурс]. – Режим доступа: https://developer.apple.com/documentation/, свободный. (Дата обращения: 29.09.2019 г.)

[5] Microsoft «rdtsc» [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/cpp/intrinsics/rdtsc?view=vs-2019, свободный. (Дата обращения: 29.09.2019 г.)