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 №5

По курсу: «Анализ алгоритмов»

Тема: «Конвейерная обработка данных»

Студент: Пронин А. С.

Группа: ИУ7-52Б

Преподаватель: Волкова Л. Л.

Оценка: _____

Москва

2021

Содержание

Введение

Цель работы – изучить муравьиный алгоритм на материале решения задачи коммивояжёра.

Задачи работы:

- описать методы решения;
- описать реализацию, реализовать метод;
- выбрать класс данных, составить набор данных;
- провести параметризацию метода на основании муравьиного алгоритма для выбранного класса данных;
- провести сравнительный анализ двух методов;
- дать рекомендации о применимости метода решения задачи коммивояжера на основе муравьиного алгоритма.

1 Аналитический раздел

Задача коммивояжёра — одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город.

Задача коммивояжёра относится к числу транс вычислительных: уже при относительно небольшом числе городов (66 и более) она не может быть решена методом перебора вариантов никакими теоретически мыслимыми компьютерами за время, меньшее нескольких миллиардов лет.

Муравьиный алгоритм — один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах. Суть подхода заключается в анализе и использовании модели поведения муравьёв, ищущих пути от колонии к источнику питания, и представляет собой метаэвристическую оптимизацию.

1.1 Описание задачи

Задача коммивояжёра заключается в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, совокупный критерий и тому подобное) и соответствующие матрицы расстояний, стоимости и тому подобного. Как правило, указывается, что маршрут должен проходить через каждый город только один раз — в таком случае выбор осуществляется среди гамильтоновых циклов.

Дано:

- M — число городов;
- D — матрица смежности.

Найти:

- Минимальный маршрут, проходящий через все города только 1 раз.

  STOPPED HERE!!!

1.2 Описание алгоритма "Рекурсивный полный перебор"

Полный перебор — метод решения математических задач. Относится к классу методов поиска решения исчерпыванием всевозможных вариантов. Сложность полного перебора зависит от количества всех возможных решений задачи. Если пространство решений очень велико, то полный перебор может не дать результатов в течение нескольких лет или даже столетий. Полный перебор гарантировано дает идеальное решение, так как гарантируется, что каждый вариант будет рассмотрен.

Однако на практике этот алгоритм не применяется, так как чаще необходимо получить, возможно, не самое лучшее решение, но за минимальное время.

1.3 Описание алгоритма "Муравьиный алгоритм"

Идея муравьиной оптимизации — моделирование поведения муравьев, связанного с их способностью быстро находить кратчайший путь. При своем движении муравей помечают путь феромоном, и эта информация используется другими муравьями для выбора пути. Это элементарное правило поведения и определяет способность муравьев находить новый путь, если старый оказывается недоступным.

По сравнению с точными методами, например динамическим программированием или методом ветвей и границ, муравьиный алгоритм находит близкие к оптимуму решения за значительно меньшее время даже для задач небольшой размерности ($n > 20$). Время оптимизации муравьиным алгоритмом является полиномиальной функцией от размерности

$O(t, n^2, m)$ **?**, тогда как для точных методов зависимость экспоненциальная.

Каждый муравей выходит из своего города и, пройдя по своему маршруту, подсчитывает стоимость пройденного маршрута. Вероятность перехода из вершины i в вершину j определяется по формуле (**??**).

$$P_{ij,k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum\limits_{l \in J_{i,k}} [\tau_{il}(t)]^{\alpha} \cdot \eta_{il}]^{\beta}} & , j \in J_{i,k}; \\ 0 & , j \notin J_{i,k} \end{cases} \tag{1.1}$$

где

$\eta_{i,j}$ – величина, обратная расстоянию от города i до j;

$\tau_{i,j}$ – количество феромонов на ребре ij;

β – параметр влияния длины пути;

α – параметр влияния феромона.

После завершения маршрута каждый муравей k откладывает на ребре (i, j) такое количество феромона (формула (**??**)):

$$\Delta\tau_{i,j}^{k} = \begin{cases} \frac{Q}{L_k} & \text{Если k-ый муравей прошел по ребру ij;} \\ 0 & \text{Иначе} \end{cases} \tag{1.2}$$

где

Q – количество феромона, переносимого муравьем;

L_k – стоимость маршрута k-го муравья

Для исследования всего пространства решений необходимо обеспечить испарение феромона — уменьшение во времени количества отложенного на предыдущих итерациях феромона. После окончания условного дня наступает условная ночь, в течение которой феромон испаряется с ребер с коэффициентом ρ. Правило обновления феромона примет вид в соответствии с формулой (**??**):

$$\tau_{i,j}(t+1) = (1-\rho)\tau_{i,j}(t) \ + \ \Delta\tau_{i,j}(t), \tag{1.3}$$

где

$$\rho_{i,j} - \text{доля феромона, который испарится;}$$
$$\tau_{i,j}(t) - \text{количество феромона на дуге ij;}$$
$$\Delta\tau_{i,j}(t) - \text{количество отложенного феромона}$$
$$t - \text{номер текущего дня}$$
$$\tau_{i,j}(t) = \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}(t)$$
$$m - \text{количество мурьавьев}$$

1.4 Вывод

В данном разделе была описана суть задачи коммивояжера и описаны алгоритмы, которые будут использоваться в данной работе.

2 Конструкторский раздел

В данном разделе представлены схемы алгоритмов рекурсивного полного перебора и муравьинного алгоритма.

2.1 Схемы алгоритмов

Схема алгоритма рекурсивного полного перебора представлена на рисунке **??**. Схема муравьинного алгоритма представлена на рисунке **??**.

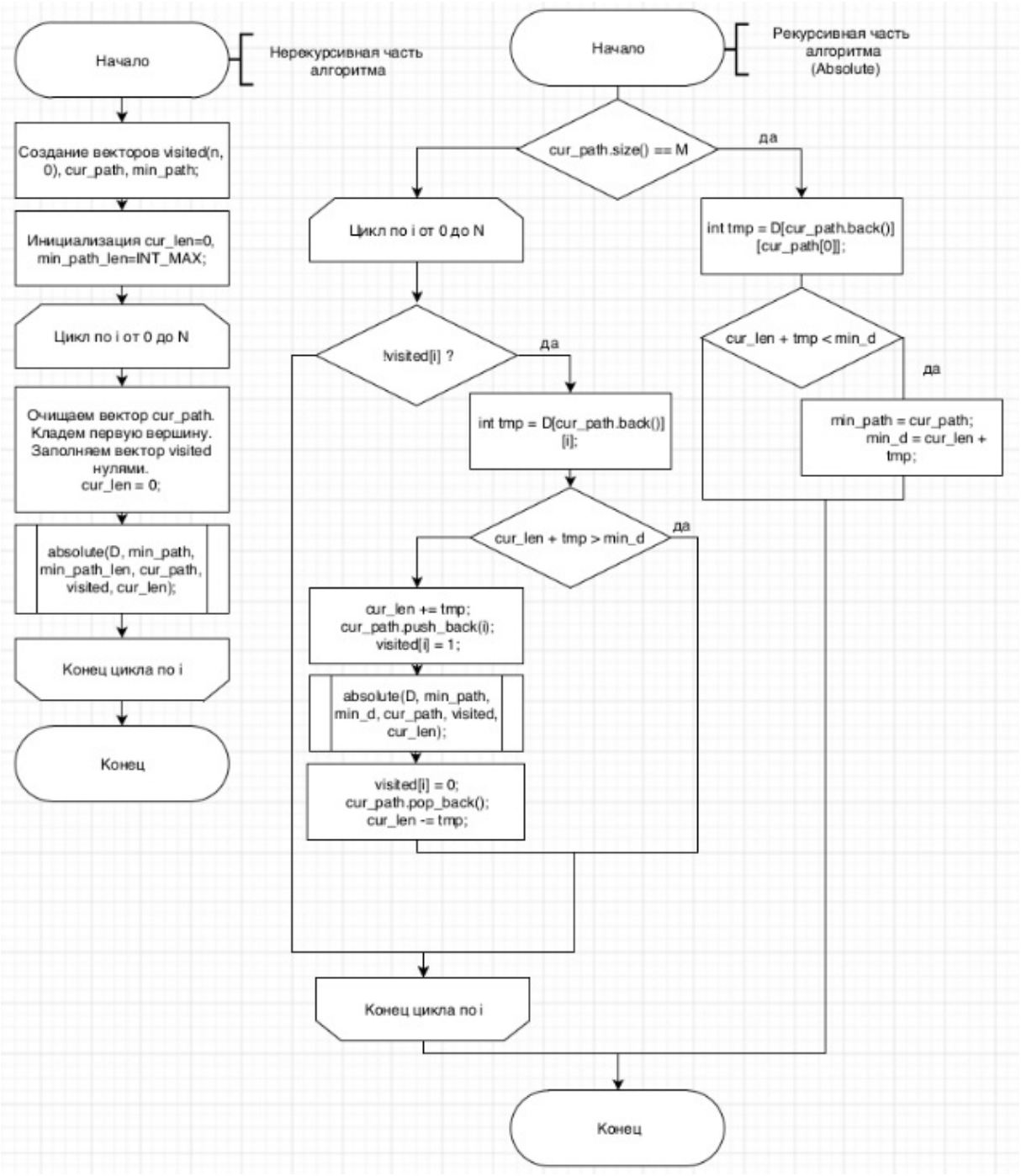

Рис. 2.1: Схема алгоритма рекурсивного полного перебора

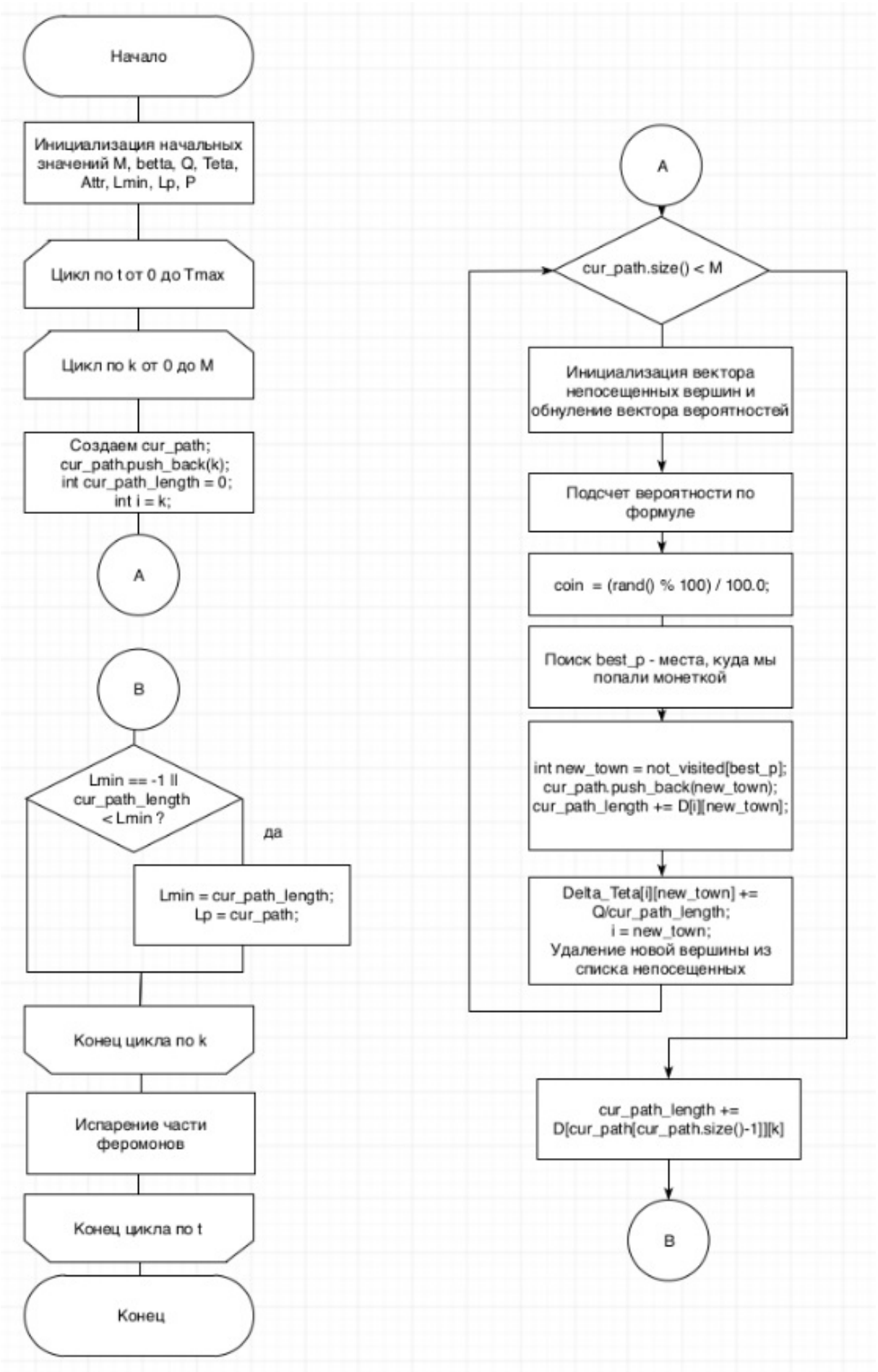

Рис. 2.2: Схема муравьинного алгоритма

2.2 Вывод

В данном разделе были разработаны схемы алгоритмов рекурсивного полного перебора и муравьинного алгоритма.

3 Технологический раздел

В данном разеделе представлены требование к ПО, выбор инструментов для реализации и оценки алгоритмов, а также листинги полученного кода.

3.1 Требование к ПО

К программе предъявляется ряд требований:

- на вход программа получает количество вершин и матрицу смежностей;
- на выходе должен быть получен минимальный путь, проходящий через все вершины графа.

3.2 Выбор инструментов

Поскольку наиболее освоенным языком для разработчика является c++, для реализации поставленной задачи был выбран именно он, т.к. таким образом работа будет проделана наиболее быстро и качественно.

Соответственно для компиляции кода будет использоваться компилятор G++.

Чтобы оценить время выполнения программы будет замерятся реальное время, т.к. таким образом можно будет сравнить реализации алгоритмов без и с использованием параллельных вычислений. Для замера реального времени работы программы используется функция clock() т.к. программа тестируется на компьютере с установленной ОС Windows. ?

Кроме этого, необходимо отключить оптимизации компилятора для более честного сравнения алгоритмов. В моём случае это делается с помощью ключа $-O0$ т.к. используется компилятор G++. ?

3.3 Реализация алгоритмов

На листингах ??-?? представлены реализации алгоритма рекурсивного полного перебора и муравьинного алгоритма.

Листинг 3.1: Реализация полного перебора, часть 1

```
pair<int, vector<int> > absolute_find(Matrix D)
{
    int n = D.rows();
    vector<bool> visited(n, 0);
    vector<int> cur_path;
    vector<int> min_path;
    int cur_len = 0;
    int min_path_len = INT_MAX;
    for (int i = 0; i < n; i++)
    {
        cur_path.clear();
        cur_path.push_back(i);
        fill(visited.begin(), visited.end(), 0);
        visited[i] = 1;
        cur_len = 0;
        absolute(D, min_path, min_path_len, cur_path, visited, cur_len);
    }
    return pair<int, vector<int>>(min_path_len, min_path);
}
```

Листинг 3.2: Реализация полного перебора, часть 2

```
void absolute(Matrix &D, vector<int> &min_path, int &min_d, vector<int> &cur_path,
    vector<bool> &visited, int &cur_len) {
    size_t M = D.cols();
    if (cur_path.size() == M) {
        ss++;
        int tmp = D[cur_path.back()][cur_path[0]];
        if (cur_len + tmp < min_d)
        {
            min_path = cur_path;
            min_d = cur_len + tmp;
        }
        return;
    }
    for (size_t i = 0; i < M; i++) {
        if (!visited[i])
        {
           int tmp = D[cur_path.back()][i];
           if (cur_len + tmp > min_d)
               continue;
           cur_len += tmp;
           cur_path.push_back(i);
           visited[i] = 1;
           absolute(D, min_path, min_d, cur_path, visited, cur_len);
           visited[i] = 0;
           cur_path.pop_back();
           cur_len -= tmp;
        }
    }
}
```

Листинг 3.3: Реализация муравьиного алгоритма

```
pair<int, vector<int>> Ant(Matrix D, const int Tmax, const double alpha, const double
    ro){
    const int teta_start = 10;
    const int teta_min = 5;
    const size_t M = D.cols(); //количество вершин
    const double Q = D.avg()*M; //примерное значение длины пути
    const double betta = A_B_CONST - alpha;

    Matrix Attr(M, M); //привлекательность
    get_attractiveness(Attr, D);
    Matrix Teta(M, M, teta_start); //феромоны
    Matrix Delta_Teta(M, M); //феромоны на тек шаге
    int Lmin = -1; //длина кратчайшего маршрута
    vector<int> Lp; // кратчайший маршрут
    vector<double> P(M, 0.0); //вероятности

    double coin;
    // цикл по времени жизни колонии
    for (int t = 0; t < Tmax; t++) {
        Delta_Teta.zero();
        //цикл по всем муравьям
        for (size_t k = 0; k < M; k++) {
            vector<int> cur_path;
            cur_path.push_back(k);
            int cur_path_length = 0;
            int i = k;
            //строим маршрут
            while (cur_path.size() < M) {
                vector<int> not_visited;
                find_not_visited(not_visited,cur_path,M);
                for (size_t j = 0; j < M ; j++){
                    P[j] = 0.0;
                }
                // посчитаем вероятности
                for (size_t j = 0; j < not_visited.size(); j++) {
                    if (D[i][not_visited[j]]) {
                        double sum = 0;
                        for (auto n: not_visited) {
                            sum += pow(Teta[i][n], alpha) * pow(Attr[i][n], betta);
                        }
                        P[j] = pow(Teta[i][not_visited[j]],
                            alpha)*pow(Attr[i][not_visited[j]], betta)/sum;
                    }
                    else {
                        P[j] = 0;
                    }
                }
```

Листинг 3.4: Реализация муравьиного алгоритма

```
                //подбросим монетку
                coin = (rand() % 100) / 100.0;
                // подсчитаем куда мы попали
                int best_p = 0;
                double sum_p = 0;
                for (size_t s = 0; s < M; s++) {
                    sum_p += P[s];
                    if(coin < sum_p)
                    {
                        best_p = s;
                        break;
                    }
                }
                // добавим новый город в путь
                int new_town = not_visited[best_p];
                cur_path.push_back(new_town);
                cur_path_length += D[i][new_town];
                //обновим феромон на этом участке
                Delta_Teta[i][new_town] += Q/cur_path_length;
                i = new_town; // дальше продолжим путь из этого города
                not_visited.erase(not_visited.begin()+best_p);
            }
            // конец построения маршрута
            // осталось только добавить путь от последнего города к начальному
            // это завершит цикл
            cur_path_length += D[cur_path[cur_path.size()-1]][k];
            // проверим не каратчайший ли это путь
            if (Lmin == -1 || cur_path_length < Lmin){
                Lmin = cur_path_length;
                Lp = cur_path;
            }
        }
        // конец цикла по муравьям
        //теперь чатсь феромона должна испариться
        for (size_t ii = 0; ii < M; ii++) {
            for (size_t jj = 0; jj < M; jj++) {
                Teta[ii][jj] = Teta[ii][jj] * (1.0 - ro) + Delta_Teta[ii][jj];
                if (Teta[ii][jj] < teta_min)
                {
                    Teta[ii][jj] = teta_min;
                }
            }
        }
    }
    // конец цикла по времени
    return pair<int, vector<int>>(Lmin, Lp);
}
```

3.4 Тестирование

STOOPED HERE! Для проверки написанных алгоритмов были подготовлены следующие тесты:

- проверка реузльтата обработки строки "test lol 23323232 s sss ll 2332323322 sls sll lls"
- проверка реузльтата обработки строки "test lol 23323232 s sss ll 233232332 sls sll lls"
- проверка реузльтата обработки строки "te str"

Для подготовленных тестов ожидаются следующие результаты соответсвтенно:

- "lol";
- "233232332";
- "No polinoms".

На рисунке **??** приведены результаты тестирования. Тестирование проводилось по методу чёрного ящика.

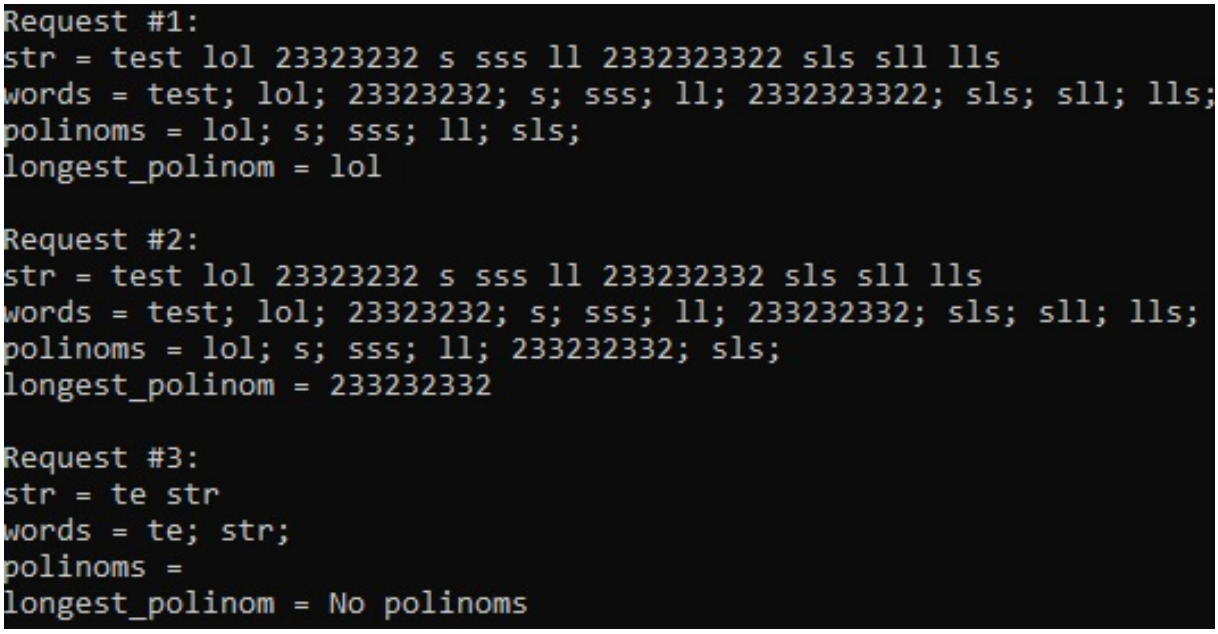

Рис. 3.1: Результаты тестирования

Как видно, все тесты пройдены.

3.5 Вывод

В данном разделе были выдвинуты требования к ПО, выбраны инструменты реализации выбранных алгоритмов, представлены листинги реализованных алгоритмов, а также проведено тестирование.

4 Исследовательский раздел

В данном разделе представлены технические характеристики компьютера, используемого для тестирования и экспериментов, и результат работы программы.

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором были проведены эксперименты при помощи разработанного ПО:

- операционная система: Windows 10 (64-разрядная);
- оперативная память: 32 GB;
- процессор: Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz;
- количество ядер: 4;
- количество потоков: 8.

4.2 Результат работы программы

На рисунке **??** представлены результаты работы программы для обработки 10 заявок, каждая из которых содержит строку из 10000 слов, состоящих из 1-10 букв.

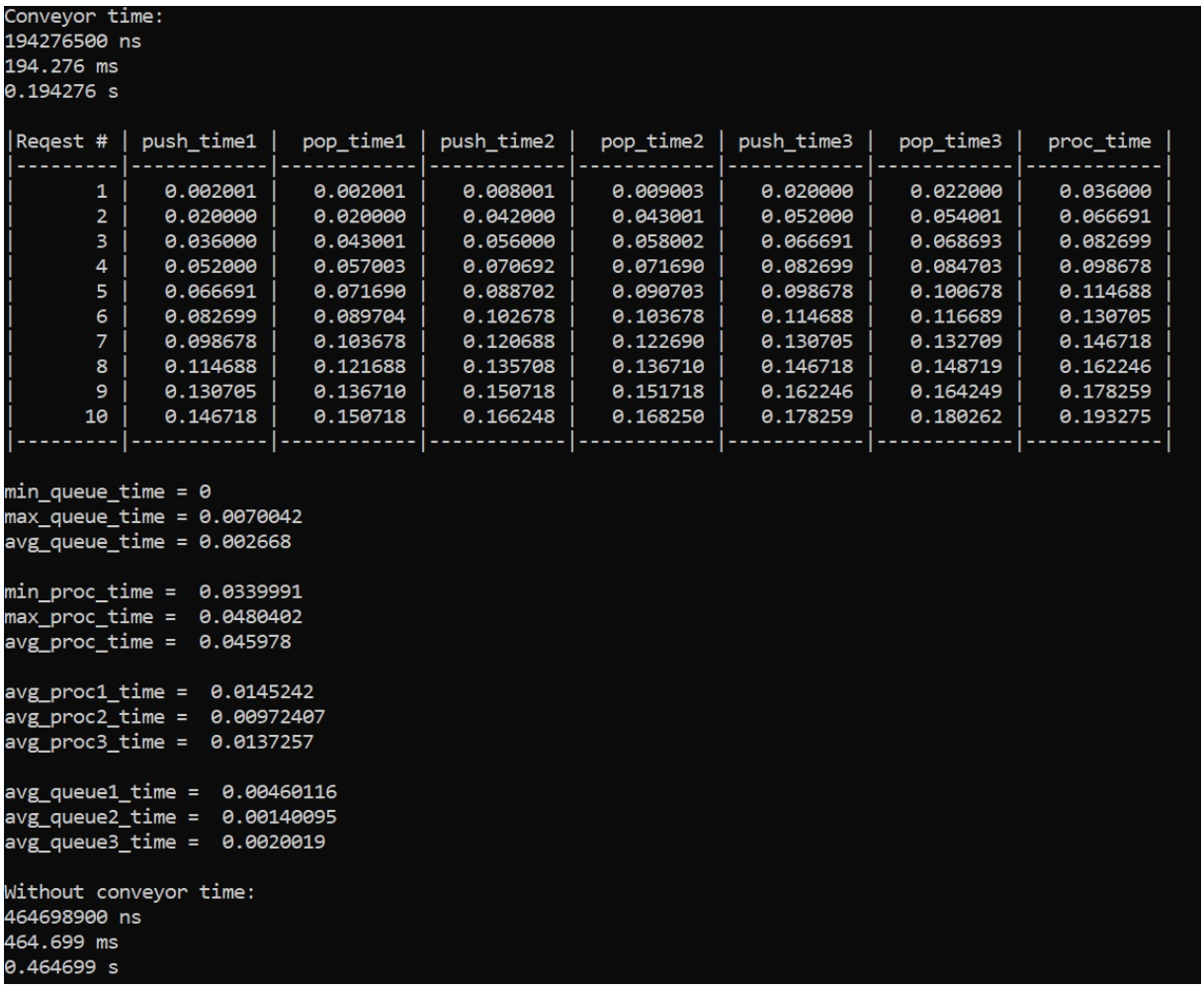

Рис. 4.1: Результат работы программы

min_queue_time - минимальное время простоя в очереди.
max_queue_time - максимальное время простоя в очереди.
avg_queue_time - среднее время простоя в очереди.
min_proc_time - минимальное время обработки заявки.
max_proc_time - максимальное время обработки заявки.
avg_proc_time - среднее время обработки заявки.
avg_procX_time - среднее время обработки для X-ой ленты.
avg_queueX_time - среднее время время простоя в X-ой очереди.

Из результатов следует, что больше всего времени в среднем тратится на разбиение строки на слова (1-я лента конвейера), меньше всего – на поиск полиндромов среди них (2-я лента конвейера). Соответственно, время простоя в первой очереди — наибольшее, а во второй — наименьшее. Также, реализация с конвейером быстрее справилась с заявками, чем реализация с последовательной обработкой.

4.3 Вывод

По итогу иследования выяснилось, что разработанный алгоритм работает верно, то есть находит самый длинный полиндром в строке. Кроме этого был проведён анализ и сделан вывод по логу программы.

Заключение

По итогу проделанной работы была достигнута цель – получен навык организации асинхронной передачи данных между потоками на примере конвейерной обработки информации.

Также были решены все поставленные задачи, а именно:

- выбраны и описаны методы обработки данных, которые сопоставлены методам конвейера;
- реализована конвейерная система, а также сформирован лог событий с указанием времени их происхождения;
- произведено сравнение реализации с конвейером и без.

Список использованных источников

MSDN – библиотека <clock> [Электронный ресурс] // Техническая документация Майкрософт. URL: https://docs.microsoft.com/ru-ru/cpp/c-runtime-library/reference/clock?view=msvc-160&viewFallbackFrom=vs-2019 (дата обращения: 13.12.21)

Как применить настройки оптимизации GCC в QT? [Электронный ресурс] // Блог ПерСа. URL: http://blog.kislenko.net/show.php?id=1991 (дата обращения: 07.12.21)