



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 Ву Хай Данг

---

Группа ИУ7и-52Б

---

Оценка (баллы)

---

Преподаватель Волкова Л. Л., Строганов Д.В.

---

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Описание конвейерной обработки данных . . . . .	4
1.2 Описание алгоритмов . . . . .	4
<b>2 Конструкторская часть</b>	<b>6</b>
2.1 Разработка алгоритмов . . . . .	6
2.2 Требования к программному обеспечению . . . . .	13
2.3 Классы эквивалентности . . . . .	13
<b>3 Технологическая часть</b>	<b>14</b>
3.1 Средства реализации . . . . .	14
3.2 Реализация алгоритмов . . . . .	14
3.3 Функциональные тесты . . . . .	21
<b>4 Исследовательская часть</b>	<b>22</b>
4.1 Технические характеристики устройства . . . . .	22
4.2 Демонстрация работы программы . . . . .	22
4.3 Время выполнения алгоритмов . . . . .	23
4.4 Вывод . . . . .	23
<b>Заключение</b>	<b>25</b>
<b>Список использованных источников</b>	<b>26</b>

# Введение

Для увеличения скорости выполнения программ используют параллельные вычисления. Конвейерная обработка данных является популярным приемом при работе с параллельностью. Она позволяет на каждой следующей «линии» конвейера использовать данные, полученные с предыдущего этапа. [1]

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (эксплуатация параллелизма на уровне инструкций). [2]

Целью данной лабораторной работы является изучение принципов конвейерной обработки данных.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- исследовать основы конвейерной обработки данных;
- привести схемы алгоритмов, используемых для конвейерной и линейной обработок данных;
- определить средства программной реализации;
- провести модульное тестирование;
- провести сравнительный анализ времени работы алгоритмов;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе.

# 1 Аналитическая часть

В этом разделе будет представлено описание сути конвейрной обработки данных и используемых алгоритмов.

## 1.1 Описание конвейрной обработки данных

Конвейер [2] — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций), технология, используемая при разработке компьютеров и других цифровых электронных устройств.

Конвейрную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Такая обработка данных в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые лентами, и выделении для каждой из них отдельного блока аппаратуры. Так, обработку любой машинной команды можно разделить на несколько этапов (лент), организовав передачу данных от одного этапа к следующему.

## 1.2 Описание алгоритмов

В данной лабораторной работе на основе конвейрной обработки данных будет поиск подстроки в строке методом полного перебора. В качестве алгоритмов на каждую из трех лент были выбраны следующие действия.

- Поиск всех вхождений подстроки в строку.
- Поиск всех вхождений развернутой подстроки в строку.
- Запись ответов в единый файл.

## Вывод

В этом разделе было рассмотрено понятие конвейрной обработки данных, а также описанно алгоритмы для обработки строки на каждой из трех лент конвейера.

## 2 Конструкторская часть

В данном разделе будут приведены схемы конвейерной и линейной реализаций алгоритма поиска подстроки в строке.

### 2.1 Разработка алгоритмов

На рис. 2.1 – 2.6 приведены схемы линейной и конвейерной реализаций алгоритма поиска подстроки в строке, схема трёх лент для конвейерной обработки строки, а также схемы реализаций этапов обработки строки.

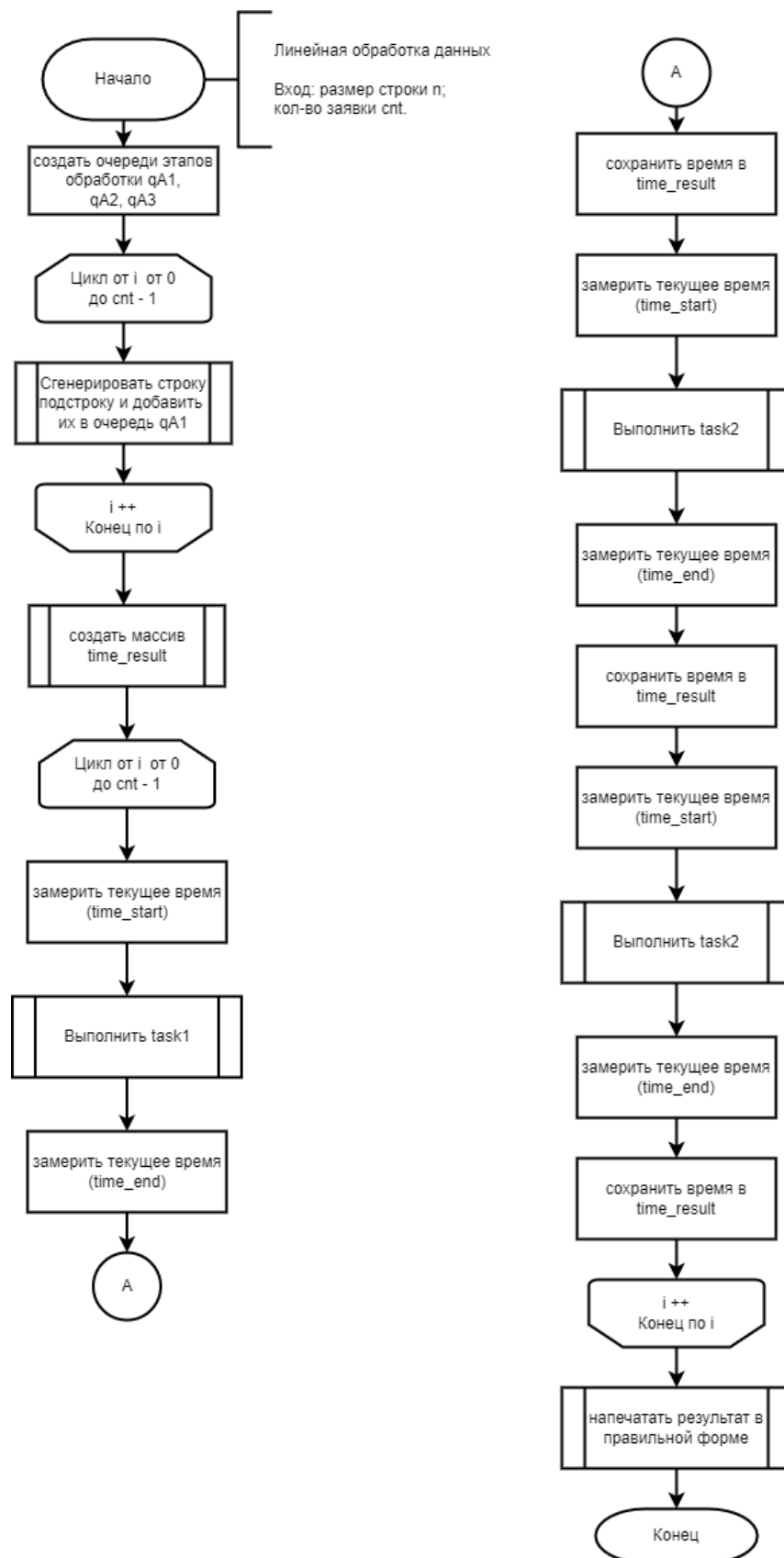


Рисунок 2.1 – Схема алгоритма линейной обработки строки

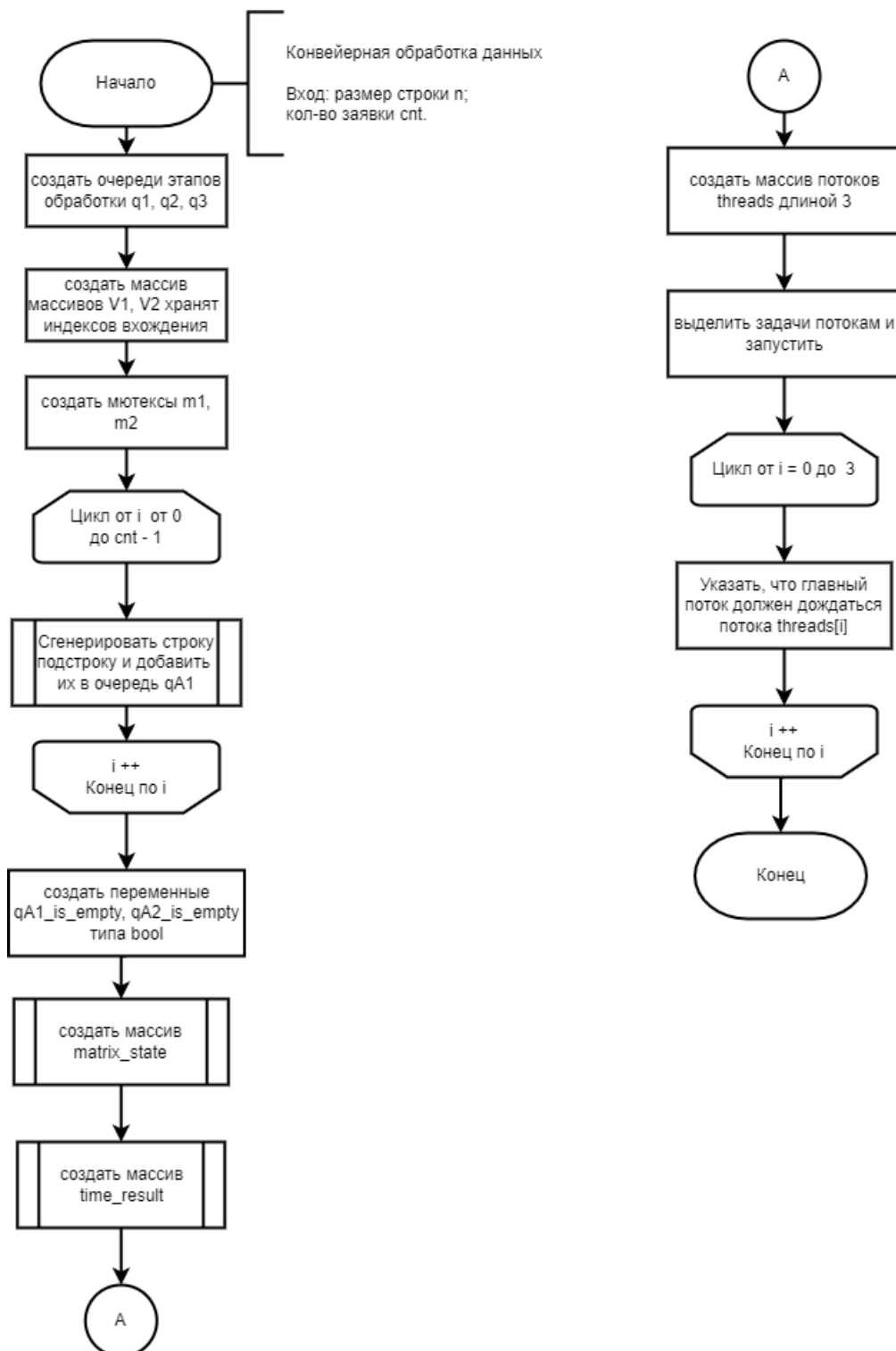


Рисунок 2.2 – Схема алгоритма конвейерной обработки строки



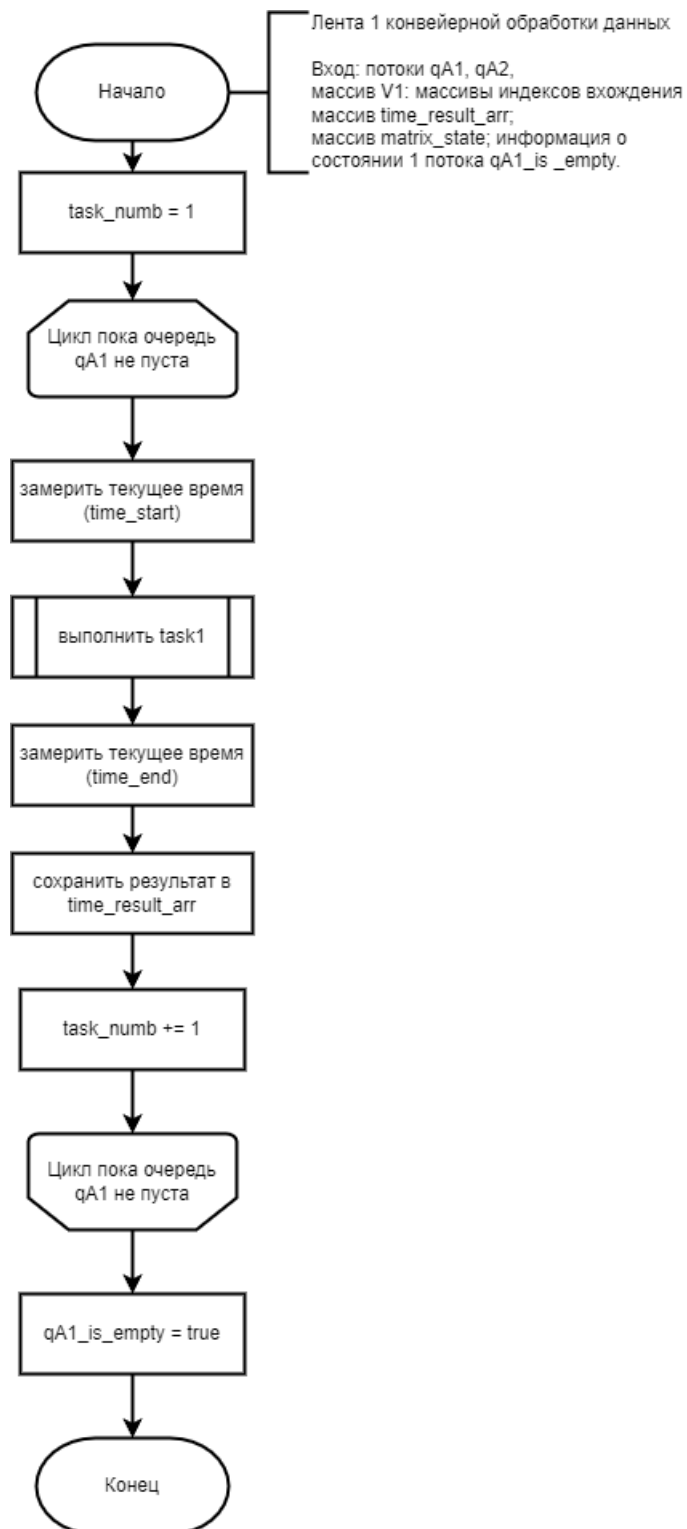


Рисунок 2.3 – Схема 1-ой ленты конвейерной обработки строки

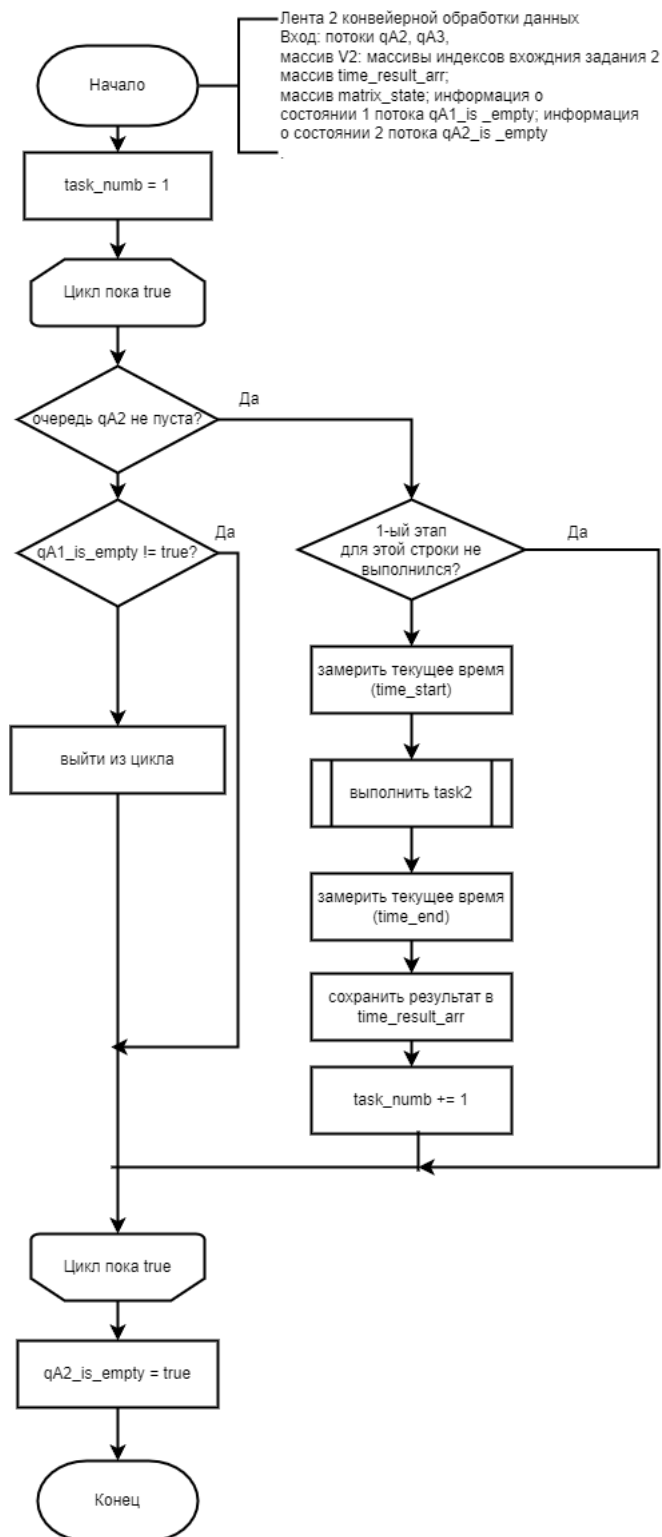


Рисунок 2.4 – Схема 2-ой ленты конвейерной обработки строки

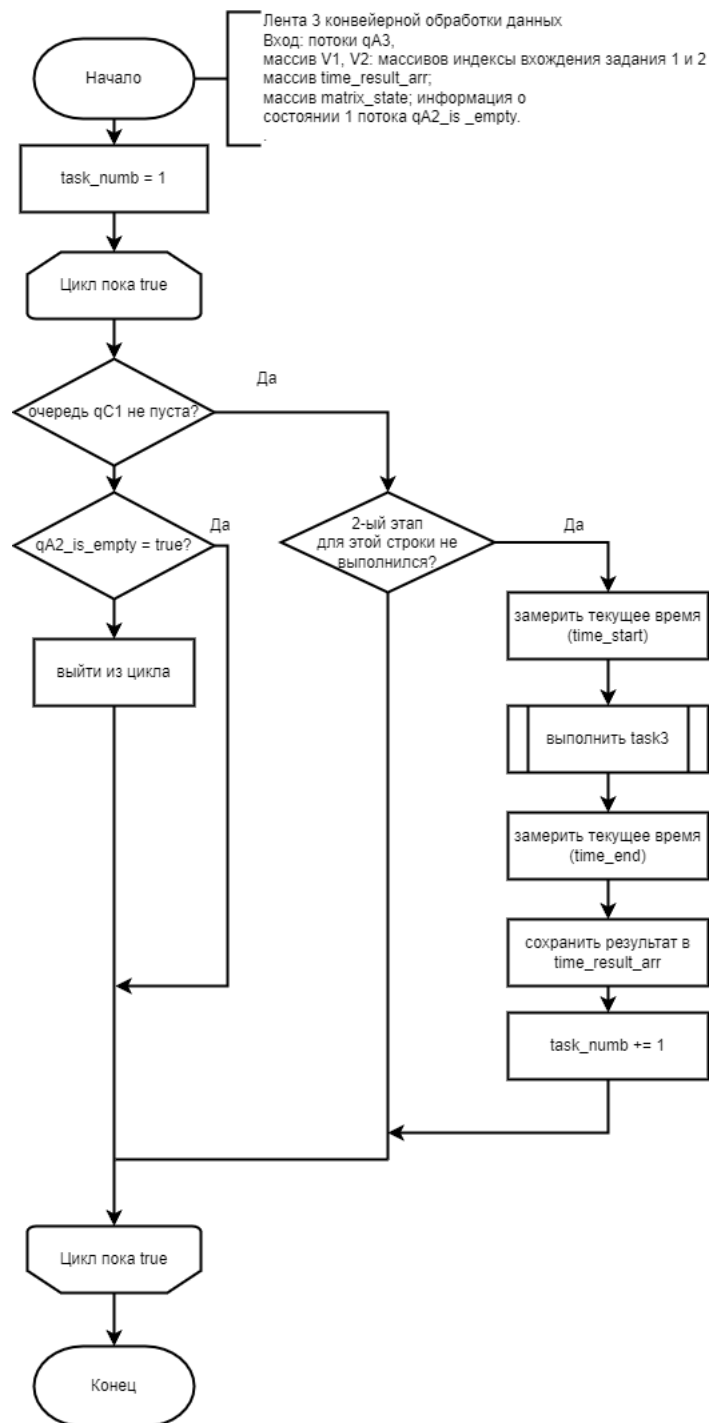


Рисунок 2.5 – Схема 3-ей ленты конвейерной обработки строки

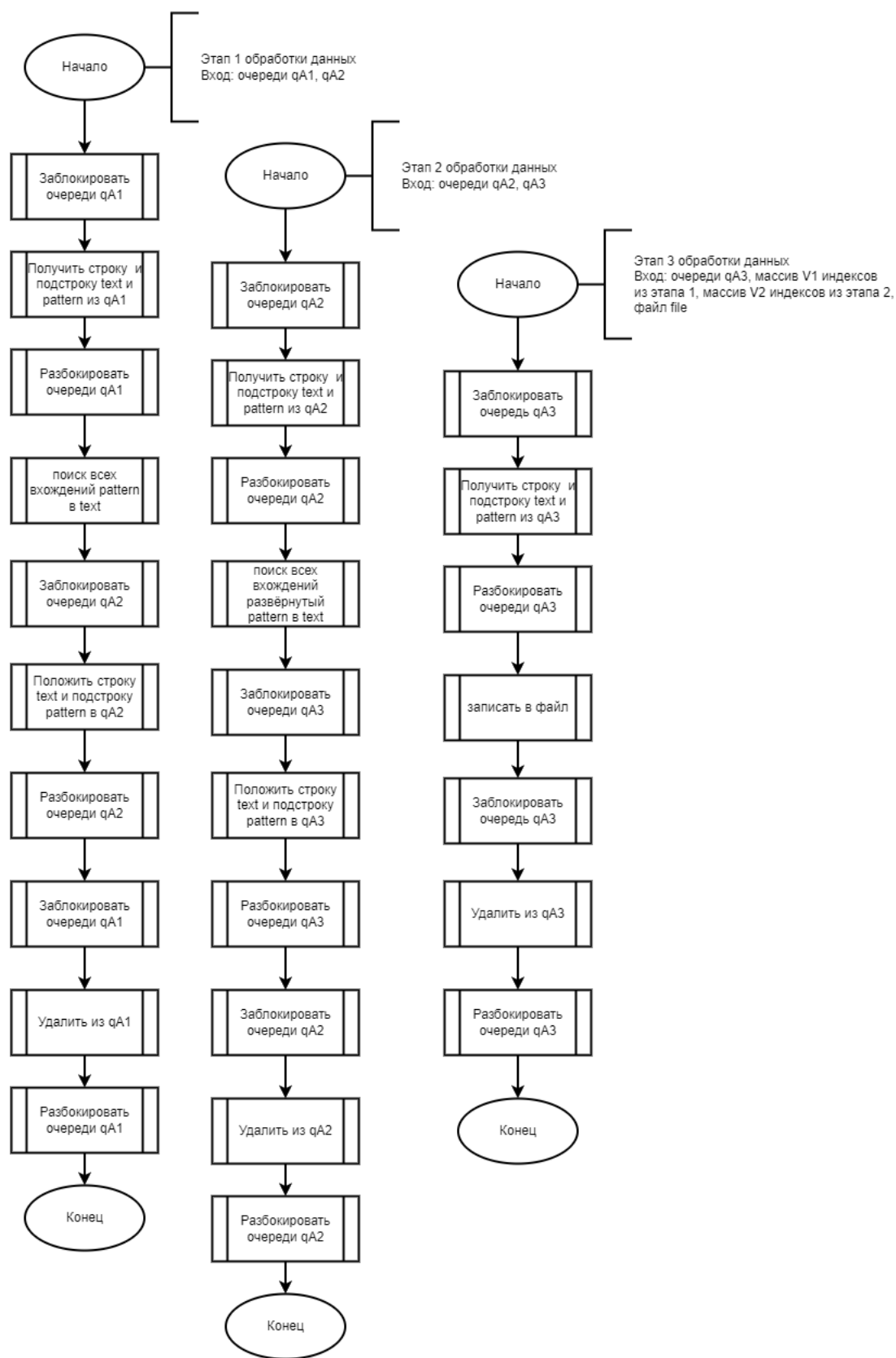


Рисунок 2.6 – Схема реализаций этапов обработки строки

## 2.2 Требования к программному обеспечению

Входные данные задаются размер строки и количество строк, которое должно быть больше 0.

Выходные данные — табличка с размерами строки, количествами строк, номерами этапов (лент) её обработки, временем начала обработки текущей строки на текущей ленте, временем окончания обработки текущей строки на текущей ленте.

## 2.3 Классы эквивалентности

Выделенные классы эквивалентности для тестирования:

- размер строки  $\leq 0$ ;
- размер строки не является целым числом;
- количество заявок  $\leq 0$ ;
- номер команды  $< 0$  или  $> 3$ ;
- номер команды не является целым числом;

## Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых методов обработки строки (конвейерного и линейного), приведены требования к программному обеспечению и выделены классы эквивалентности для тестирования.

## 3 Технологическая часть

В данном разделе будут приведены средства реализации, реализация алгоритма, а также функциональные тесты.

### 3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *C++* [3], так как он предоставляет весь необходимый функционал для выполнения работы. Для замера времени работы использовалась функция *std::chrono::system\_clock::now()* [4].

### 3.2 Реализация алгоритмов

В листингах 3.1 - 3.8 представлены функции для конвейерного и ленивого алгоритмов обработки строки.

Листинг 3.1 – Функция алгоритма конвейерной обработки строки

```
1 void parallel(int size, int cnt, bool is_count)
2 {
3     queue<strings_t> qA1;
4     queue<strings_t> qA2;
5     queue<strings_t> qA3;
6     vector<vector<int>> V1;
7     vector<vector<int>> V2;
8     bool qA1_is_empty = false;
9     bool qA2_is_empty = false;
10    mutex m1;
11    mutex m2;
12    mutex time_mutex;
13
14    for (int i = 0; i < cnt; i++)
15    {
16        strings_t strings;
17        strings.text = random_text(size);
18        strings.pattern = random_text(size / 4);
```

```

19         qA1.push(strings);
20     }
21
22     vector<strings_state_t> state(cnt);
23     for (int i = 0; i < cnt; i++)
24     {
25         strings_state_t tmp_state;
26         tmp_state.stage_1 = false;
27         tmp_state.stage_2 = false;
28         tmp_state.stage_3 = false;
29         state[i] = tmp_state;
30     }
31     chrono::time_point<std::chrono::system_clock> time_begin =
        chrono::system_clock::now();
32     vector<res_time_t> time_result_arr;
33     init_time_result_arr(time_result_arr, time_begin, cnt, 3);
34     thread threads[3];
35     threads[0] = thread(parallel_stage_1, ref(time_mutex), ref(m1),
        ref(qA1), ref(qA2), ref(V1), ref(state),
        ref(time_result_arr), ref(qA1_is_empty));
36     threads[1] = thread(parallel_stage_2, ref(time_mutex), ref(m1),
        ref(m2), ref(qA2), ref(qA3), ref(V2), ref(state),
        ref(time_result_arr), ref(qA1_is_empty), ref(qA2_is_empty));
37     threads[2] = thread(parallel_stage_3, ref(time_mutex), ref(m2),
        ref(qA3), ref(V1), ref(V2), ref(state),
        ref(time_result_arr), ref(qA2_is_empty));
38
39     for (int i = 0; i < 3; i++)
40     threads[i].join();
41     if (is_count)
42     printf("#####%4d#####|#####%4d#####|###%.6f###\n",
43     size, cnt, time_result_arr[cnt - 1].end);
44     else
45     print_res_time(time_result_arr, cnt * 3);
46 }

```

### Листинг 3.2 – Функция алгоритма линейной обработки строки

```

1 void liner(int size, int cnt, bool is_count)
2 {
3     queue<strings_t> qA1;
4     queue<strings_t> qA2;
5     queue<strings_t> qA3;
6     vector<int> tmpV1;
7     vector<int> tmpV2;
8     mutex m1;
9     mutex m2;
10
11     std::chrono::time_point<std::chrono::system_clock> time_start,
        time_end,
12     time_begin = std::chrono::system_clock::now();
13
14     std::vector<res_time_t> time_result_arr;
15     init_time_result_arr(time_result_arr, time_begin, cnt, 3);
16
17     for (int i = 0; i < cnt; i++)
18     {
19         strings_t strings;
20         strings.text = random_text(size);
21         strings.pattern = random_text(size / 4);
22
23         qA1.push(strings);
24     }
25
26     for (int i = 0; i < cnt; i++)
27     {
28         time_start = chrono::system_clock::now();
29         tmpV1 = task1(ref(m1), ref(qA1), ref(qA2));
30         time_end = std::chrono::system_clock::now();
31
32         save_result(time_result_arr, time_start, time_end,
            time_result_arr[0].time_begin, i + 1, 1);
33
34         time_start = chrono::system_clock::now();
35         tmpV2 = task2(ref(m1), ref(m2), ref(qA2), ref(qA3));
36         time_end = std::chrono::system_clock::now();
37
38         save_result(time_result_arr, time_start, time_end,

```



```

39         time_result_arr[0].time_begin, i + 1, 2);
40     time_start = chrono::system_clock::now();
41     task3(ref(m2), ref(qA3), ref(tmpV1), ref(tmpV2), Myfile);
42     time_end = std::chrono::system_clock::now();
43
44     save_result(time_result_arr, time_start, time_end,
45                time_result_arr[0].time_begin, i + 1, 3);
46 }
47 if (is_count)
48 {
49     printf("uuuuuu%4duuuuuu|uuuuuu%4duuuuuu|uuuu%.6fuu\n",
50           size, cnt, time_result_arr[cnt - 1].end);
51 }
52 else
53 {
54     print_res_time(time_result_arr, cnt * 3);
55 }
56 }

```

Листинг 3.3 – Функция 1-ой ленты конвейерной обработки строки

```

1 void parallel_stage_1(mutex& time_mutex, mutex& m1,
2   queue<strings_t>& qA1, queue<strings_t>& qA2,
3   vector<vector<int>>& V1, vector<strings_state_t>& state,
4   vector<res_time_t>& time_result_arr, bool & qA1_is_empty)
5 {
6     chrono::time_point<chrono::system_clock> time_start, time_end;
7     int task_num = 1;
8     while (!qA1.empty())
9     {
10         time_start = chrono::system_clock::now();
11         vector<int> tmp = task1(m1, qA1, qA2);
12         time_end = std::chrono::system_clock::now();
13         time_mutex.lock();
14         save_result(time_result_arr, time_start, time_end,
15                    time_result_arr[0].time_begin, task_num, 1);
16         time_mutex.unlock();
17         V1.push_back(tmp);
18         state[task_num - 1].stage_1 = true;
19         task_num++;

```

```

16     }
17     qA1_is_empty = true;
18 }

```

Листинг 3.4 – Функция 2-ой ленты конвейерной обработки строки

```

1 void parallel_stage_2(mutex& time_mutex, mutex& m1, mutex& m2,
    queue<strings_t>& qA2, queue<strings_t>& qA3,
    vector<vector<int>>& V2, vector<strings_state_t>& state,
    vector<res_time_t>& time_result_arr, bool &qA1_is_empty, bool &
    qA2_is_empty)
2 {
3     chrono::time_point<chrono::system_clock> time_start, time_end;
4     int task_num = 1;
5     while (1)
6     {
7         if (qA2.empty() == false)
8         {
9             if (state[task_num - 1].stage_1 == true)
10            {
11                time_start = chrono::system_clock::now();
12                vector<int> tmp = task2(m1, m2, qA2, qA3);
13                time_end = std::chrono::system_clock::now();
14                time_mutex.lock();
15                save_result(time_result_arr, time_start, time_end,
                    time_result_arr[0].time_begin, task_num, 2);
16                time_mutex.unlock();
17                V2.push_back(tmp);
18                state[task_num - 1].stage_2 = true;
19                task_num++;
20            }
21        }
22        else if (qA1_is_empty)
23            break;
24    }
25    qA2_is_empty = true;
26 }

```

### Листинг 3.5 – Функция 3-ей ленты конвейерной обработки строки

```

1 void parallel_stage_3(mutex& time_mutex, mutex& m2,
  queue<strings_t>& qA3, vector<vector<int>>& V1,
  vector<vector<int>>& V2, vector<strings_state_t>& state,
  vector<res_time_t>& time_result_arr, bool& qA2_is_empty)
2 {
3     chrono::time_point<chrono::system_clock> time_start, time_end;
4     int task_num = 1;
5     while (1)
6     {
7         if (qA3.empty() == false)
8         {
9             if (state[task_num - 1].stage_2 == true &&
              state[task_num - 1].stage_1 == true)
10            {
11                time_start = chrono::system_clock::now();
12                task3(m2, qA3, V1[task_num - 1], V2[task_num - 1],
                  Myfile);
13                time_end = std::chrono::system_clock::now();
14                time_mutex.lock();
15                save_result(time_result_arr, time_start, time_end,
                  time_result_arr[0].time_begin, task_num, 3);
16                time_mutex.unlock();
17                state[task_num - 1].stage_3 = true;
18                task_num++;
19            }
20        }
21        else if (qA2_is_empty)
22            break;
23    }
24 }

```

Листинг 3.6 – Функция реализации 1-ого этапа обработки строки

```
1 vector<int> task1(mutex& m1, queue<strings_t>& qA1,  
    queue<strings_t>& qA2)  
2 {  
3     m1.lock();  
4     strings_t tmpStrings = qA1.front();  
5     m1.unlock();  
6     vector<int> tmpV = algoS(tmpStrings.text, tmpStrings.pattern);  
7  
8     m1.lock();  
9     qA2.push(tmpStrings);  
10    m1.unlock();  
11  
12    qA1.pop();  
13    return tmpV;  
14 }
```

Листинг 3.7 – Функция реализации 2-ого этапа обработки строки

```
1 vector<int> task2(mutex& m1, mutex& m2, queue<strings_t>& qA2,  
    queue<strings_t>& qA3)  
2 {  
3  
4     m1.lock();  
5     strings_t tmpStrings = qA2.front();  
6     m1.unlock();  
7  
8     vector<int> tmpV = algoS(tmpStrings.text,  
        reverse_string(tmpStrings.pattern));  
9  
10    m2.lock();  
11    qA3.push(tmpStrings);  
12    m2.unlock();  
13    m1.lock();  
14    qA2.pop();  
15    m1.unlock();  
16    return tmpV;  
17 }
```

### Листинг 3.8 – Функция реализации 3-его этапа обработки строки

```
1 void task3(mutex& m2, queue<strings_t>& qA3, vector<int>& V1,  
    vector<int>& V2, ofstream& file)  
2 {  
3     m2.lock();  
4     strings_t tmpStrings = qA3.front();  
5     m2.unlock();  
6  
7     write_to_file(file, tmpStrings.pattern, tmpStrings.text, V1,  
        V2);  
8  
9     m2.lock();  
10    qA3.pop();  
11    m2.unlock();  
12 }
```

## 3.3 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для конвейерного и линейного алгоритмов обработки строки. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Размер строки	Кол-во заявок	Алгоритм	Ожидаемый результат
-10	10	Конвейерный	Сообщение об ошибке
10	-10	Конвейерный	Сообщение об ошибке
0	10	Конвейерный	Сообщение об ошибке
k	10	Конвейерный	Сообщение об ошибке
100	20	Конвейерный	Вывод результ. таблички
100	20	Линейный	Вывод результ. таблички

## Вывод

В данном разделе были разработаны алгоритмы для конвейерного и линейного алгоритмов обработки матриц, проведено тестирование, описаны средства реализации.

## 4 Исследовательская часть

### 4.1 Технические характеристики устройства

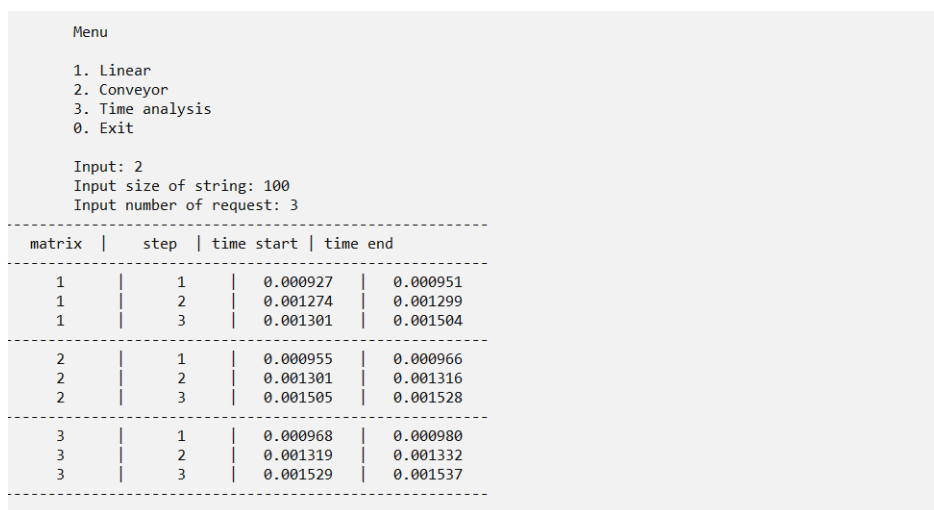
Ниже представлены характеристики компьютера, на котором проводилось тестирование программы:

- операционная система Windows 10 Домашняя;
- оперативная память 12 Гб;
- процессор Intel(R) Core(TM) i7-9750H CPU @ 2.6 ГГц.

Во время тестирования ноутбук был подключен к сети электропитания. Процессор был загружен на 19%, оперативная память – на 50%.

### 4.2 Демонстрация работы программы

На рисунках 4.1 – 4.2 приведены примеры работы программы конвейерной реализаций и линейной реализаций.



```
Menu
1. Linear
2. Conveyor
3. Time analysis
0. Exit

Input: 2
Input size of string: 100
Input number of request: 3
```

matrix	step	time start	time end
1	1	0.000927	0.000951
1	2	0.001274	0.001299
1	3	0.001301	0.001504
2	1	0.000955	0.000966
2	2	0.001301	0.001316
2	3	0.001505	0.001528
3	1	0.000968	0.000980
3	2	0.001319	0.001332
3	3	0.001529	0.001537

Рисунок 4.1 – Пример работы программы (конвейерная реализация)

Menu				
1. Linear				
2. Conveyor				
3. Time analysis				
0. Exit				
Input: 1				
Input size of string: 100				
Input number of request: 3				
matrix	step	time start	time end	
1	1	0.000064	0.000084	
1	2	0.000084	0.000099	
1	3	0.000099	0.000193	
2	1	0.000193	0.000207	
2	2	0.000207	0.000220	
2	3	0.000220	0.000229	
3	1	0.000230	0.000241	
3	2	0.000242	0.000254	
3	3	0.000254	0.000263	

Рисунок 4.2 – Пример работы программы (линейная реализация)

### 4.3 Время выполнения алгоритмов

Чтобы получить достаточно точное значение, производилось усреднение времени. Количество запусков замера процессорного времени 1000 раз.

В таблице 4.1 приведены результаты замеров времени работы реализаций линейного и конвейерного алгоритмов с одним размером строки.

Таблица 4.1 – Результаты замеров времени

Размер строки	Кол-во строки	Линейный	Конвейерный
100	50	0.002075	0.002269
100	100	0.004045	0.001729
100	200	0.008096	0.003848
100	400	0.016149	0.005938
100	800	0.032413	0.013172
100	1600	0.064927	0.022478

В таблице 4.2 приведены результаты замеров времени работы реализаций линейного и конвейерного алгоритмов с одним количеством строки.

Таблица 4.2 – Результаты замеров времени

Размер строки	Кол-во строки	Линейный	Конвейерный
20	100	0.00211	0.001348
40	100	0.00251	0.001348
80	100	0.003892	0.001441
160	100	0.005379	0.001733
320	100	0.008571	0.002195

## 4.4 Вывод

В этом разделе были указаны технические характеристики машины, на которой происходило сравнение времени работы алгоритмов обработки строк для конвейерной и линейной реализаций.

В результате замеров времени было установлено, что конвейерная реализация обработки лучше линейной при большом количестве строк (в 3 раза при 400 строки, 800 и 1600). Так же конвейерная обработка показала себя лучше при увеличении размеров обрабатываемых строк (в 5 раза при размере строки 160 и 320). Значит при большом количестве обрабатываемых строк, а также при обработке строки большого размера стоит использовать конвейерную реализацию обработки, а не линейную.



## Заключение

В результате исследований можно сделать вывод о том, что при большом количестве обрабатываемых строк, а так же при обработке строки большого размера стоит использовать конвейерную реализацию обработки, а не линейную (при обработке 1600 строк с размерами 100 конвейерная быстрее в 3 раза, а при обработке 100 строк с размерами 320 быстрее в 4 раза).

В ходе выполнения данной лабораторной работы были решены следующие задачи:

- изучены основы конвейерной обработки данных;
- применены изученные основы для реализации конвейерной обработки строк;
- определены средства программной реализации;
- проведены модульное тестирование;
- проведены сравнительный анализ линейной и конвейерной реализаций обработки строк;
- описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе.

Поставленная цель была достигнута.

## Список использованных источников

1. Е. Н. Акимова Параллельные вычисления, 2-е изд.: Екатеринбург: Издательство Уральского университета, 2023.
2. Конвейерная организация [Электронный ресурс]. — URL: [http://www.citforum.mstu.edu.ru/hardware/svk/glava\\_5.shtml](http://www.citforum.mstu.edu.ru/hardware/svk/glava_5.shtml).
3. C++ — Типизированный язык программирования / Хабр [Электронный ресурс]. — URL: <https://habr.com/ru/hub/cpp/>.
4. std::chrono::system\_clock::now - cppreference.com [Электронный ресурс]. — URL: [https://en.cppreference.com/w/cpp/chrono/system\\_clock/now](https://en.cppreference.com/w/cpp/chrono/system_clock/now).
5. Intel [Электронный ресурс]. — URL: <https://www.intel.ru/content/www/ru/ru/products/details/processors/core/i5.html>.