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 дисциплине "Анализ алгоритмов"

Тема Конвейерная обработка данных

Студент Хамзина Р. Р.

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

Существуют задачи, в которых разные алгоритмы обрабатывают один и тот же набор данных друг за другом. При этом, может стоять задача обработки большого объема данных. Для ускорения решения таких вычислительных задач используется конвейерная обработка.

Идея конвейерной обработки [1] заключается в выделении отдельных этапов выполнения общей операции. Каждый этап, выполнив свою работу, передает результат следующему, одновременно принимая новую порцию входных данных. Отдельный этап называют лентой. При таком способе организации вычислений увеличивается скорость обработки за счет совмещения прежде разнесенных во времени операций.

В многопоточном программировании конвейерная обработка реализуется следующим образом: под каждую ленту конвейера выделяется отдельный поток. Выделенные потоки работают асинхронно.

Целью данной лабораторной работы является организация асинхронного взаимодействия потоков на примере конвейерной обработки данных. Для достижения поставленной цели требуется выполнить следующие задачи:

- описать последовательный вариант алгоритма задачи;
- выделить алгоритмы, выполняемые на отдельных лентах конвейера;
- описать конвейерный вариант алгоритма задачи;
- привести схемы рассмотренных вариантов алгоритма;
- описать используемые типы и структуры данных;
- описать структуру разрабатываемого программного обеспечения;
- определить средства программной реализации изученных вариантов алгоритма;
- реализовать разработанные варианты алгоритма;
- провести функциональное тестирование программного обеспечения;

- провести сравнительный анализ по времени последовательной и конвейерной реализаций;
- подготовить отчет о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будут описаны последовательный и конвейерный варианты алгоритма. Также будут изучены алгоритмы, выполняемые на отдельных лентах конвейера.

1.1 Последовательный алгоритм

В некоторых задачах происходит обработка текстовых данных перед их использованием в решении. Данные проходят ряд преобразований в несколько последовательных этапов. Каждый этап реализуется при помощи алгоритма работы со строками.

Строка [2] — это последовательность ASCII или UNICODE символов. В некоторых системных языках программирования в качестве строк используются массивы символов. В этом случае с символами строк можно работать как с элементами массива.

В данной лабораторной работе будет реализована обработка строк или массивов символов, состоящая из следующих этапов:

- удаление пробелов в строке;
- перевод каждого символа строки в другой регистр;
- определение строки-палиндрома.

1.2 Алгоритмы, выполняемые на отдельных лентах

Каждый из описанных выше этапов будет выполняться на отдельной ленте.

1.2.1 Удаление пробелов в строке

Удаление пробелов в строке будет происходить следующим образом:

- каждый символ строки сравнивается с символом пробела;
- символ удаляется из строки при совпадении с пробелом.

1.2.2 Перевод символов строки в другой регистр

В данном алгоритме выполняются следующие действия для каждого символа:

- если символ в верхнем регистре, то он заменяется на символ в нижнем регистре;
- если символ в нижнем регистре, то он заменяется на символ в верхнем регистре.

1.2.3 Определение строки-палиндрома

Палиндром - это симметричный относительно своей середины набор символов. Чтобы определить, является ли строка s длины len палиндромом необходимо:

- каждый символ строки $s[i]$ сравнить с символом $s[len - i - 1]$, для $i \in [0; len/2]$;
- если текущая пара символов не совпадает, то строка не является палиндромом;
- если все символы из заданного промежутка i совпали, то строка - палиндром.

Таким образом, в последовательном алгоритме:

- строки из набора алгоритма добавляются в первую очередь;
- из первой очереди извлекается строка $s1$ и обрабатывается на первой ленте;

- строка $s1$ добавляется во вторую очередь;
- из второй очереди извлекается строка $s1$ и обрабатывается на второй ленте;
- строка $s1$ добавляется в третью очередь;
- из третьей очереди извлекается строка $s1$ и обрабатывается на третьей ленте;
- из первой очереди извлекается строка $s2$ и обрабатывается на первой ленте;
- далее каждая строка из набора последовательно повторяет цикл обработки.

1.3 Параллельный алгоритм

При помощи многопоточности можно ускорить процесс обработки строк. В этом случае для каждой ленты создается отдельный поток. Извлечение и добавление строк осуществляется потоками. Так, в параллельном алгоритме:

- первый поток извлекает строку из первой очереди, она обрабатывается на первой ленте, и поток добавляет ее во вторую очередь;
- второй поток извлекает строку из второй очереди, она обрабатывается на второй ленте, и поток добавляет ее в третью очередь;
- третий поток извлекает строку из второй очереди, и она обрабатывается на третьей ленте;
- каждый поток по завершении вновь извлекает строку из своей очереди, и цикл обработки повторяется в параллельном режиме.

1.4 Вывод

Для реализации последовательной и конвейерной обработки данных был выбран алгоритм работы со строкой. На трех лентах выполняются следующие этапы алгоритма:

- удаление пробелов в строке;
- перевод каждого символа строки в другой регистр;
- определение строки-палиндрома.

Программе, реализующей данный алгоритм, на вход будет подаваться набор строк определенной длины. Программа должна работать в рамках следующих ограничений:

- символами строк могут быть буквы латинского алфавита в верхнем и нижнем регистре и пробел;
- должно быть выдано сообщение об ошибке при вводе не числовых или неположительных длины и числа строк.

Пользователь должен иметь возможность выбора обработки строк: последовательно или при помощи конвейера. Также должны быть реализованы сравнение вариантов обработки по времени работы в зависимости от количества строк в наборе и в зависимости от длины строк с выводом результатов на экран и получение графического представления результатов сравнения. Результат данных действий пользователь должен получать при помощи меню.

2 Конструкторская часть

В данном разделе будут представлены схемы последовательной и конвейерной обработки строк. Будут описаны типы и структуры данных, используемые для реализации, а также структура разрабатываемого программного обеспечения. Кроме того, будут выделены классы эквивалентности для тестирования.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема последовательной обработки набора строк, на рисунке 2.2 - параллельного. Схемы алгоритмов работы лент конвейера показаны на рисунках 2.3-2.5. Схема алгоритма удаления пробелов в строке представлена на рисунке 2.6, перевода регистра симовлов строки в другой - на рисунке 2.7, определение строки-палиндрома - 2.8.

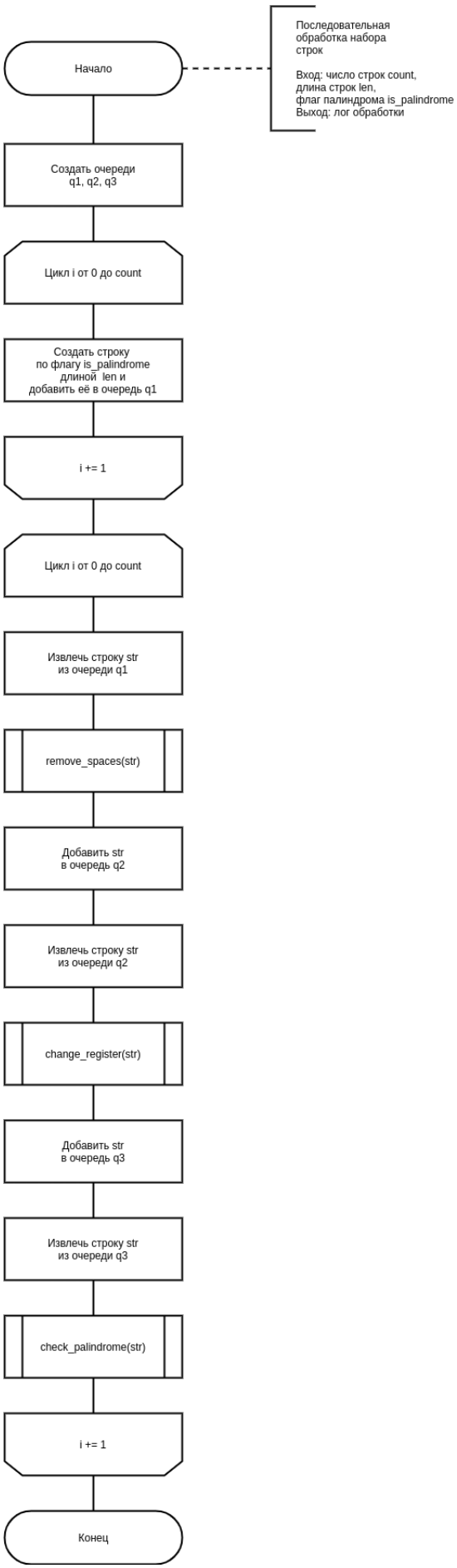

Рисунок 2.1 – Последовательная обработка набора строк

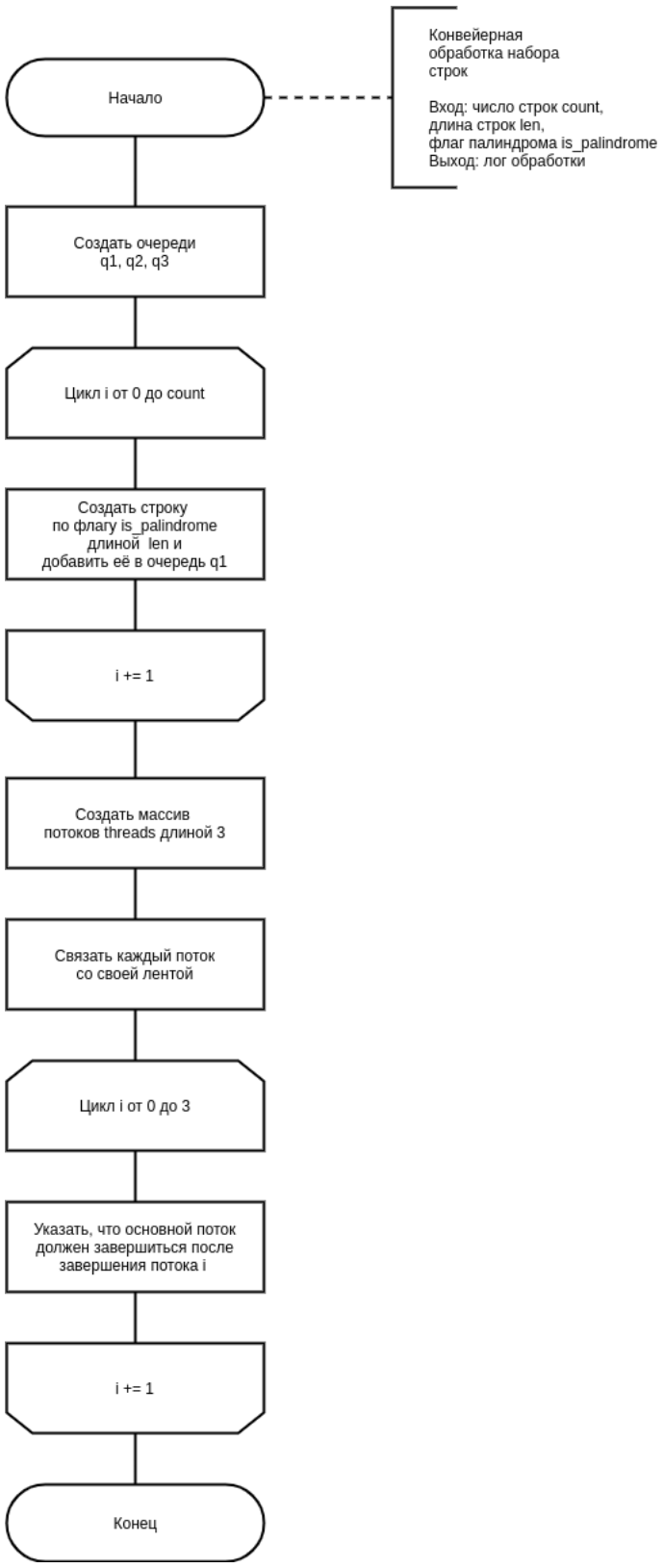

Рисунок 2.2 – Конвейерная обработка набора строк

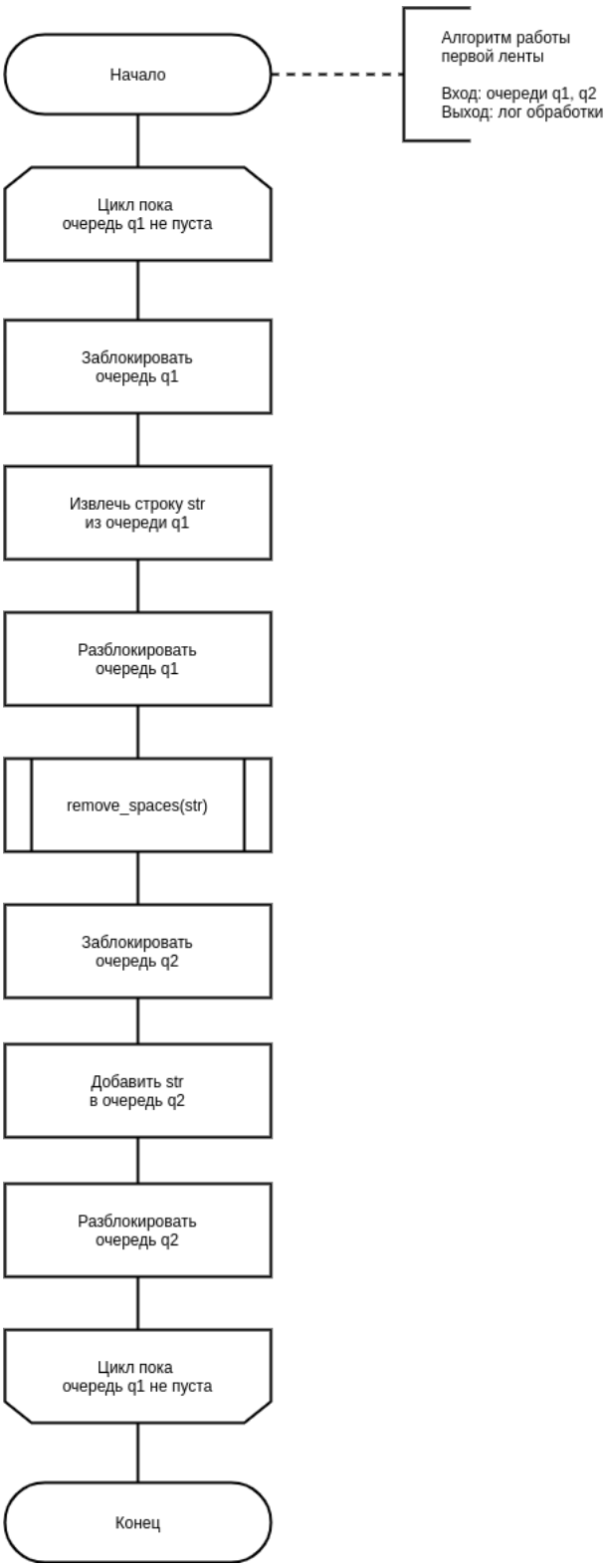

Рисунок 2.3 – Алгоритм работы первой ленты

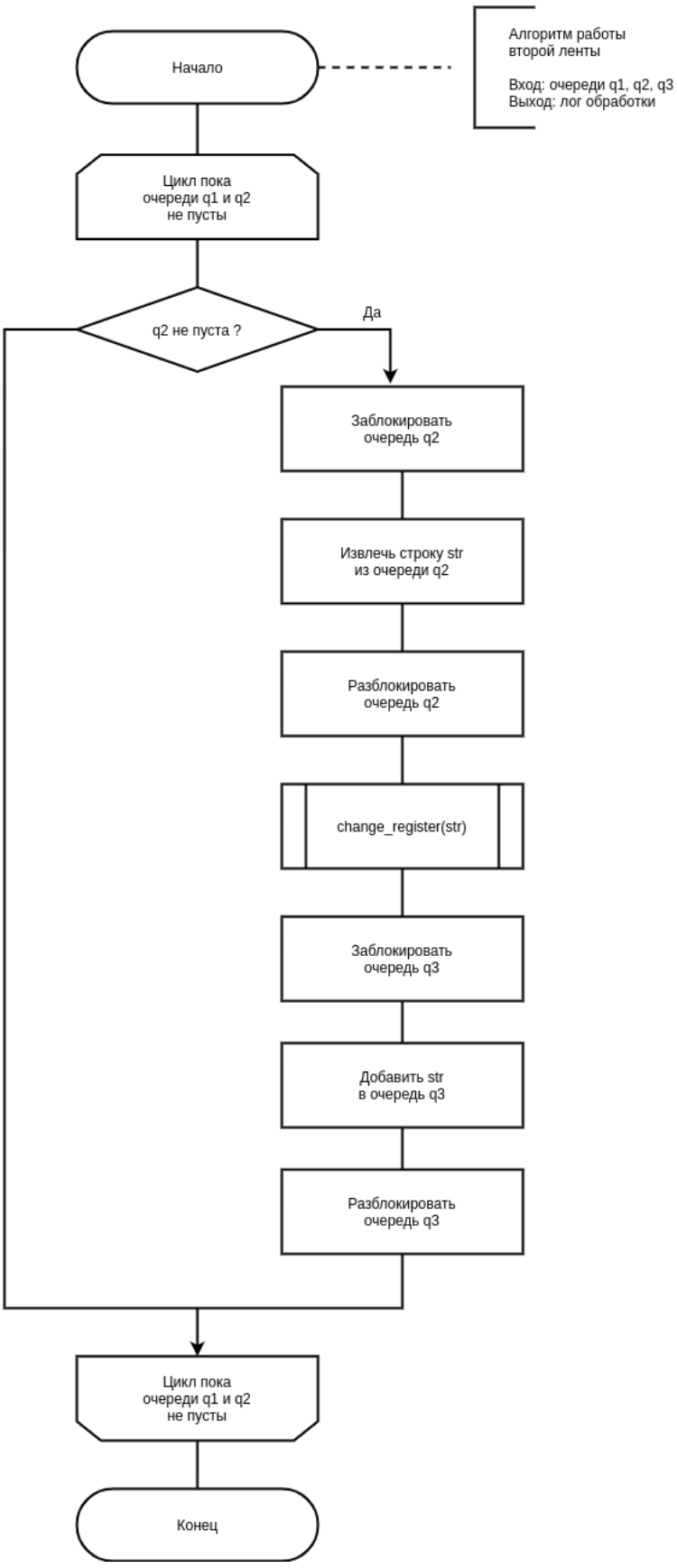

Рисунок 2.4 – Алгоритм работы второй ленты

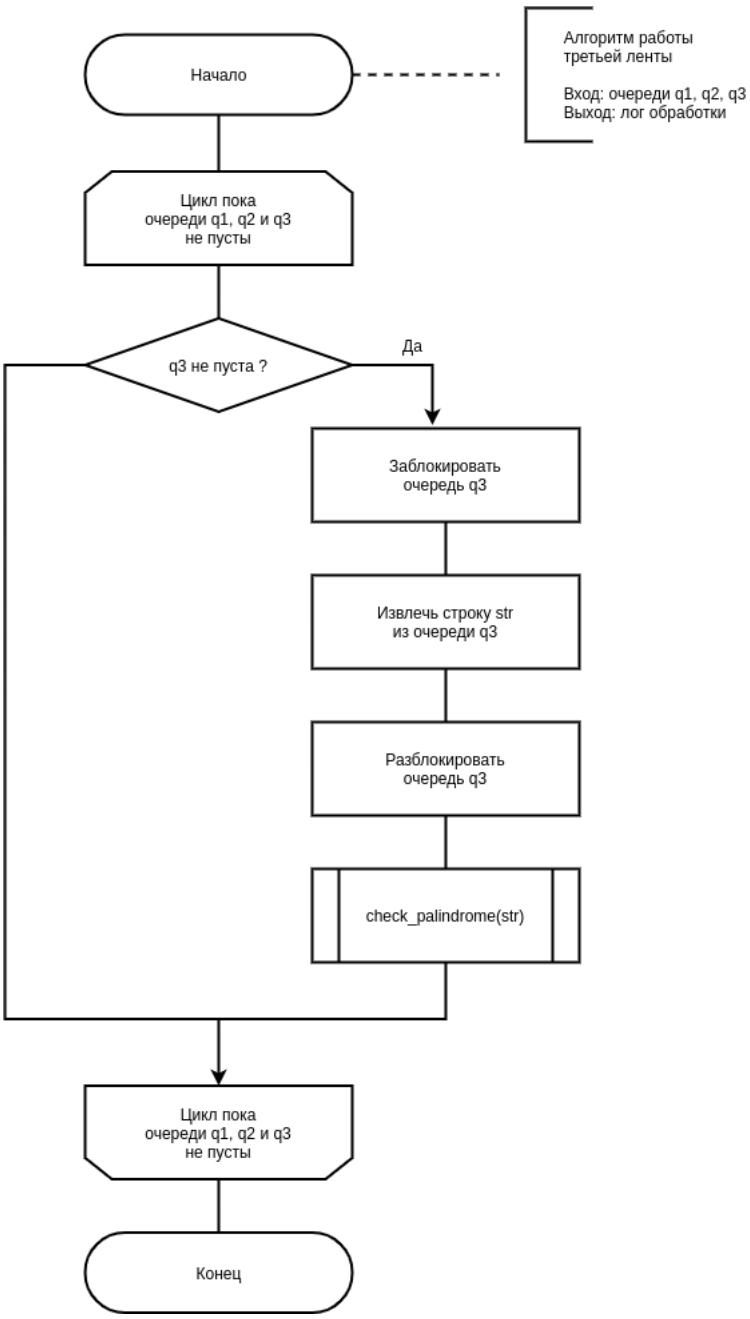

Рисунок 2.5 – Алгоритм работы третьей ленты

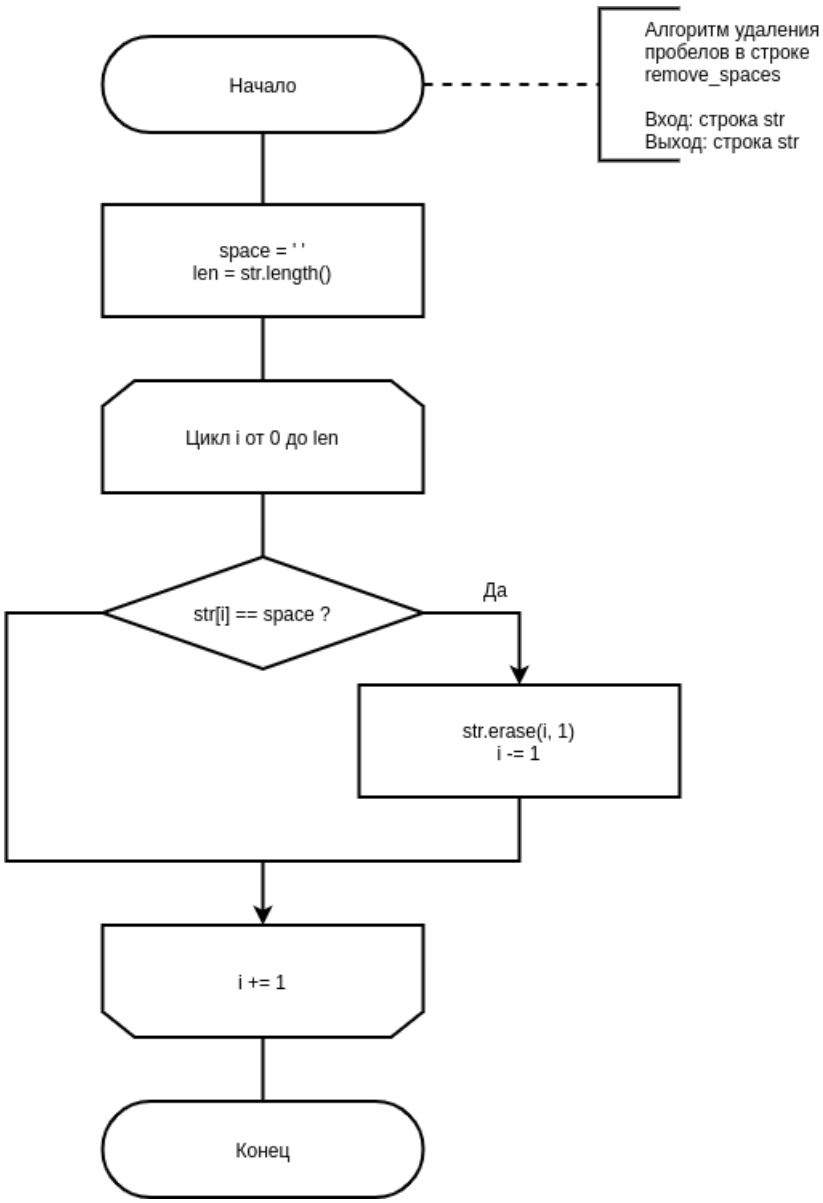

Рисунок 2.6 – Алгоритм удаления пробелов в строке

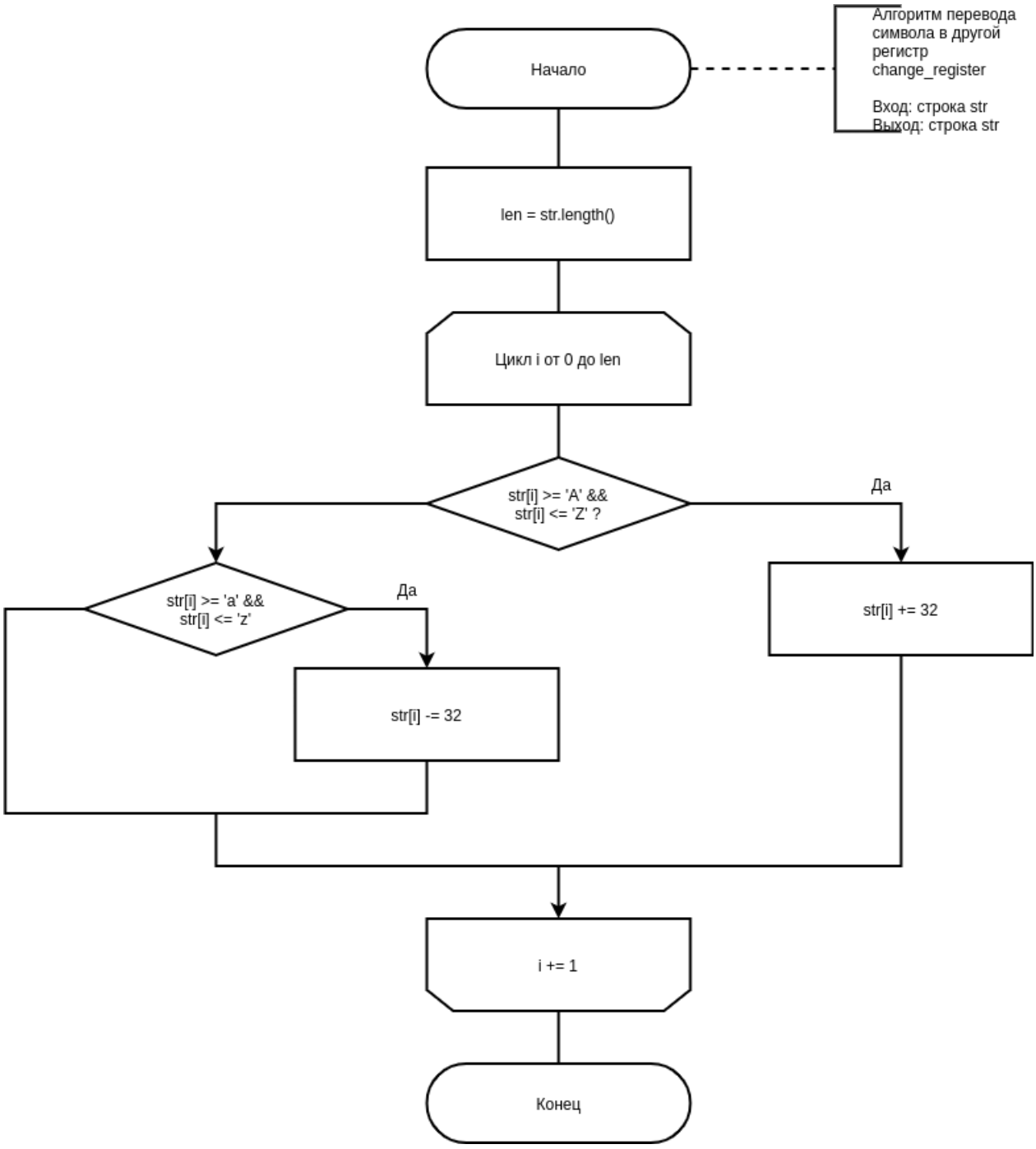

Рисунок 2.7 – Алгоритм перевода регистров символов строки

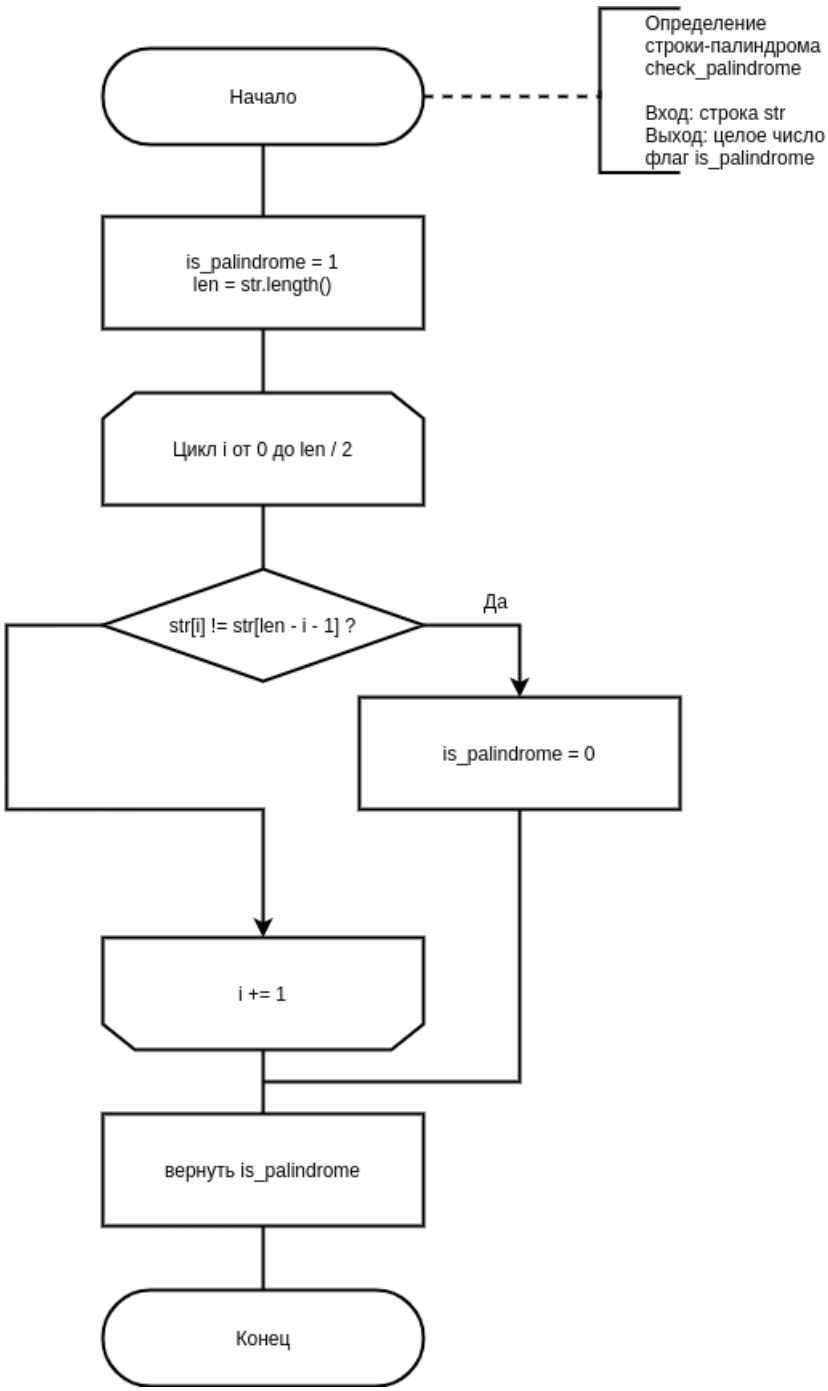

Рисунок 2.8 – Алгоритм определения строки-палиндрома

2.2 Описание используемых типов и структур данных

Для реализации набора строк будут использованы следующие типы:

- тип *int* для числа строк в наборе и для длины строки;

- $std :: string$ для строки.

Для представления очередей будет использован тип $std :: queue$, для потоков - $std :: thread$.

2.3 Структура разрабатываемого ПО

При реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Для взаимодействия с пользователем в функции $intmain(void)$ будет реализовано меню, при помощи которого будут вызываться последовательная и параллельная обработка строк и функции сравнительного анализа.

Для работы со строками будут разработаны следующие функции, принимающие на вход ссылку на строку:

- процедура создания строки, состоящей из случайных символов из допустимого набора, входными параметрами процедуры также являются длина строки и флаг для создания строки-палиндрома;
- создание графа со случайными весами ребер, выходным параметром функции является сгенерированная матрица смежности;
- процедура удаления пробелов;
- процедура перевода в другой регистр;
- функция определения строки-палиндрома.

Для организации обработки набора строк будут реализованы:

- процедуры последовательной и конвейерной обработки, у которых на входе - число строк, их длина и флаг для создания строки-палиндрома;
- процедуры работы с очередями, входными параметрами которых являются ссылки на очереди.

Для сравнительного анализа будут реализованы:

- процедуры логирования, у которых на входе - ссылка на строку и номера ленты и задачи;
- процедуры замеров времени;
- функция графического представления замеров времени, у которой на входе - массив временных значений, на выходе - его графическое представление.

2.4 Классы эквивалентности при тестировании

Для тестирования разрабатываемой программы будут выделены следующие классы эквивалентности:

- некорректная длина строки (не число или неположительное число);
- некорректный ввод числа строк (не число или неположительное число);
- корректный ввод всех параметров.

2.5 Вывод

Были представлены схемы обработки строк. Были указаны типы и структуры данных, используемые для реализации, и описана структура разрабатываемого программного обеспечения. Также были выделены классы эквивалентности для тестирования ПО.

3 Технологическая часть

В данном разделе будут указаны средства реализации, сведения о модулях программы, будут представлены листинги кода, а также функциональные тесты.

3.1 Средства реализации

Реализация данной лабораторной работы выполнялась при помощи языка программирования $C++$ [3]. Выбор ЯП обусловлен наличием инструментов для реализации принципов многопоточного программирования.

Замеры времени проводились при помощи функции $std :: chrono :: system\_clock :: now(...)$ из библиотеки *chrono* [4].

Реализация графического представления замеров времени производилась при помощи языка программирования *Python* [5], так как данный ЯП представляет простую в использовании графическую библиотеку.

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- main.cpp - главный файл программы, предоставляющий пользователю меню для выполнения основных функций;
- str.cpp и str.h - файлы, содержащие функции работы со строкой;
- conveyor.cpp и conveyor.h - файлы, содержащие функции обработки данных;
- time_test.cpp и time_test.h - файлы, содержащие функции замеров времени работы указанных алгоритмов;
- constants.h - файл, содержащий все необходимые константы;
- graph_result.py - файл, содержащий функции визуализации временных характеристик описанных алгоритмов.

3.3 Листинги кода

В листингах 3.1-3.2 представлены алгоритмы последовательной и конвейерной обработки строк. В листингах 3.3-3.5 представлены алгоритмы работы трех лент. В листингах 3.6-3.8 представлены алгоритмы работы со строками.

Листинг 3.1 – Последовательная обработка

```
1  void linear_mode(const int count, const int len,
2                   const int is_palindrome)
3  {
4      std::queue<std::string> q1;
5      std::queue<std::string> q2;
6      std::queue<std::string> q3;
7
8      for ( int i =0; i < count; ++i )
9      {
10         std::string new_str;
11         create_str(new_str, len, is_palindrome);
12
13         q1.push(new_str);
14     }
15
16     cur_time = 0;
17
18     for ( int i = 0; i < count; ++i )
19     {
20         std::string str = q1.front();
21         linear_log(str, 1, i + 1);
22         q2.push(str);
23         q1.pop();
24
25         str = q2.front();
26         linear_log(str, 2, i + 1);
27         q3.push(str);
28         q2.pop();
29
30         str = q3.front();
31         linear_log(str, 3, i + 1);
32         q3.pop();
33     }
34 }
```

Листинг 3.2 – Конвейерная обработка

```
1  void parallel_mode(const int count, const int len,
2                     const int is_palindrome)
3  {
4      std::queue<std::string> q1;
5      std::queue<std::string> q2;
6      std::queue<std::string> q3;
7
8      for ( int i = 0; i < count; ++i )
9      {
10         std::string str;
11         create_str(str, len, is_palindrome);
12
13         q1.push(str);
14     }
15
16     for ( int i = 0; i < count + 1; ++i )
17     {
18         cur_time1.push_back(0);
19         cur_time2.push_back(0);
20         cur_time3.push_back(0);
21     }
22
23     std::vector<std::thread> threads(COUNT_THREADS);
24
25     threads[0] = std::thread(first_belt, std::ref(q1),
           std::ref(q2));
26     threads[1] = std::thread(second_belt, std::ref(q1),
           std::ref(q2), std::ref(q3));
27     threads[2] = std::thread(third_belt, std::ref(q1),
           std::ref(q2), std::ref(q3));
28
29     for ( int i = 0; i < COUNT_THREADS; ++i )
30     {
31         threads[i].join();
32     }
33 }
```

Листинг 3.3 – Алгоритм работы первой ленты

```
1  void first_belt(std::queue<std::string>& q1,
       std::queue<std::string>& q2)
2  {
3      int task = 0;
4      std::mutex m;
5
6      while ( !q1.empty())
7      {
8          m.lock();
9          std::string str = q1.front();
10         m.unlock();
11
12         parallel_log(str, 1, ++task);
13
14         m.lock();
15         q2.push(str);
16         q1.pop();
17         m.unlock();
18     }
19 }
```

Листинг 3.4 – Алгоритм работы второй ленты

```
1  void second_belt(std::queue<std::string>& q1,
       std::queue<std::string>& q2, std::queue<std::string>& q3)
2  {
3      int task = 0;
4      std::mutex m;
5
6      do
7      {
8          if ( !q2.empty() )
9          {
10             m.lock();
11             std::string str = q2.front();
12             m.unlock();
13
14             parallel_log(str, 2, ++task);
15
16             m.lock();
17             q3.push(str);
18             q2.pop();
19             m.unlock();
20         }
21     } while ( !q1.empty() || !q2.empty() );
22 }
```

Листинг 3.5 – Алгоритм работы третьей ленты

```
void third_belt(std::queue<std::string>& q1,
   std::queue<std::string>& q2, std::queue<std::string>& q3)
{
    int task = 0;
    std::mutex m;

    do
    {
        if ( !q3.empty() )
        {
            m.lock();
            std::string str = q3.front();
            m.unlock();

            parallel_log(str, 3, ++task);

            m.lock();
            q3.pop();
            m.unlock();
        }
    } while ( !q1.empty() || !q2.empty() || !q3.empty() );
}
```

Листинг 3.6 – Алгоритм удаления пробелов

```
void remove_spaces(std::string& str)
{
    const char space = ' ';
    int len = str.length();

    for ( int i = 0; i < len; ++i )
    {
        if ( str[i] == space )
        {
            str.erase(i, 1);
            i--;
        }
    }
}
```

Листинг 3.7 – Алгоритм перевода в другой регистр

```
1  void change_register(std::string& str)
2  {
3      int len = str.length();
4
5      for ( int i = 0; i < len; ++i )
6      {
7          if ( str[i] >= 'A' && str[i] <= 'Z' )
8          {
9              str[i] += DIFF;
10         }
11         else if ( str[i] >= 'a' && str[i] <= 'z' )
12         {
13             str[i] -= DIFF;
14         }
15     }
16 }
```

Листинг 3.8 – Алгоритм определения палиндрома

```
1  int check_palindrome(std::string& str)
2  {
3      int result = PALINDROME;
4      int len = str.length();
5
6      for ( int i = 0; i < len / 2; ++i )
7      {
8          if ( str[i] != str[len - i - 1] )
9          {
10             result = NOT_PALINDROME;
11             break;
12         }
13     }
14
15     return result;
16 }
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для функций, реализующих строковые алгоритмы. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Обработка	Число строк	Длина строк	Ожидаемый результат
Последовательная	0	100	Сообщение об ошибке
Конвейерная	50	-2	Сообщение об ошибке
Последовательная	50	1000	Лог обработки
Конвейерная	60	400	Лог обработки

3.5 Вывод

Были реализованы функции последовательной и конвейерной обработки набора строк. Было проведено функциональное тестирование указанных функций.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, и будет проведен сравнительный анализ реализованных алгоритмов по числу строк и по длине строк.

4.1 Технические характеристики

Тестирование проводилось на устройстве со следующими техническими характеристиками:

- операционная система: Ubuntu 20.04.1 Linux x86_64 [6];
- память : 8 GiB;
- процессор: AMD® Ryzen™ 3 3200u @ 2.6 GHz;
- 4 физических ядра, 4 логических ядра [7].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 приведен пример последовательной обработки, на рисунке 4.2 - пример конвейерной обработки.

```
МЕНЮ:
1. Последовательная обработка
2. Конвейерная обработка
3. Замер времени работы алгоритма в зависимости от длины строк
4. Замер времени работы алгоритма в зависимости от числа строк
0. Выход
Выбор: 1

Длина строк: 1000

Число строк: 7
Tape: 1  Task: 1  Start Time: 0.000000  End Time: 0.000014
Tape: 2  Task: 1  Start Time: 0.000014  End Time: 0.000048
Tape: 3  Task: 1  Start Time: 0.000048  End Time: 0.000054
Tape: 1  Task: 2  Start Time: 0.000054  End Time: 0.000062
Tape: 2  Task: 2  Start Time: 0.000062  End Time: 0.000094
Tape: 3  Task: 2  Start Time: 0.000094  End Time: 0.000100
Tape: 1  Task: 3  Start Time: 0.000100  End Time: 0.000107
Tape: 2  Task: 3  Start Time: 0.000107  End Time: 0.000143
Tape: 3  Task: 3  Start Time: 0.000143  End Time: 0.000150
Tape: 1  Task: 4  Start Time: 0.000150  End Time: 0.000161
Tape: 2  Task: 4  Start Time: 0.000161  End Time: 0.000193
Tape: 3  Task: 4  Start Time: 0.000193  End Time: 0.000200
Tape: 1  Task: 5  Start Time: 0.000200  End Time: 0.000210
Tape: 2  Task: 5  Start Time: 0.000210  End Time: 0.000243
Tape: 3  Task: 5  Start Time: 0.000243  End Time: 0.000250
Tape: 1  Task: 6  Start Time: 0.000250  End Time: 0.000261
Tape: 2  Task: 6  Start Time: 0.000261  End Time: 0.000295
Tape: 3  Task: 6  Start Time: 0.000295  End Time: 0.000302
Tape: 1  Task: 7  Start Time: 0.000302  End Time: 0.000312
Tape: 2  Task: 7  Start Time: 0.000312  End Time: 0.000344
Tape: 3  Task: 7  Start Time: 0.000344  End Time: 0.000351
```

Рисунок 4.1 – Пример последовательной обработки

```
МЕНЮ:
1. Последовательная обработка
2. Конвейерная обработка
3. Замер времени работы алгоритма в зависимости от длины строк
4. Замер времени работы алгоритма в зависимости от числа строк
0. Выход
Выбор: 2

Длина строк: 1000

Число строк: 7

Tape: 1  Task: 1  Start Time: 0.000000  End Time: 0.000011
Tape: 1  Task: 2  Start Time: 0.000011  End Time: 0.000019
Tape: 1  Task: 3  Start Time: 0.000019  End Time: 0.000028
Tape: 1  Task: 4  Start Time: 0.000028  End Time: 0.000038
Tape: 1  Task: 5  Start Time: 0.000038  End Time: 0.000047
Tape: 1  Task: 6  Start Time: 0.000047  End Time: 0.000058
Tape: 1  Task: 7  Start Time: 0.000058  End Time: 0.000070
Tape: 2  Task: 1  Start Time: 0.000011  End Time: 0.000041
Tape: 3  Task: 1  Start Time: 0.000041  End Time: 0.000049
Tape: 2  Task: 2  Start Time: 0.000041  End Time: 0.000073
Tape: 3  Task: 2  Start Time: 0.000073  End Time: 0.000081
Tape: 2  Task: 3  Start Time: 0.000073  End Time: 0.000104
Tape: 3  Task: 3  Start Time: 0.000104  End Time: 0.000113
Tape: 2  Task: 4  Start Time: 0.000104  End Time: 0.000135
Tape: 3  Task: 4  Start Time: 0.000135  End Time: 0.000143
Tape: 2  Task: 5  Start Time: 0.000135  End Time: 0.000167
Tape: 3  Task: 5  Start Time: 0.000167  End Time: 0.000175
Tape: 2  Task: 6  Start Time: 0.000167  End Time: 0.000235
Tape: 3  Task: 6  Start Time: 0.000235  End Time: 0.000243
Tape: 2  Task: 7  Start Time: 0.000235  End Time: 0.000266
Tape: 3  Task: 7  Start Time: 0.000266  End Time: 0.000273
```

Рисунок 4.2 – Пример конвейерной обработки

4.3 Время выполнения алгоритмов

Функция $std :: chrono :: system\_clock :: now(...)$ из библиотеки *chrono* ЯП $C++$ возвращает процессорное время в секундах - значение типа float.

Для замера времени необходимо получить значение времени до начала выполнения алгоритма, затем после его окончания. Чтобы получить результат, необходимо вычесть из второго значения первое.

Сравнительный анализ по времени в зависимости от числа строк проводился для строк-палиндромов, заполненных случайным образом, длиной 6000 символов. Результаты измерения времени приведены в таблице 4.1 (в с).

На рисунке 4.3 приведены графические результаты сравнения временных характеристик для различного числа потоков.

Таблица 4.1 – Результаты замеров времени в зависимости от числа строк

Число строк	Конвейерная	Последовательная
10	0.00150191	0.00208726
20	0.0026085	0.00298096
30	0.00288755	0.00376455
40	0.00439648	0.0047981
50	0.00562291	0.00612586

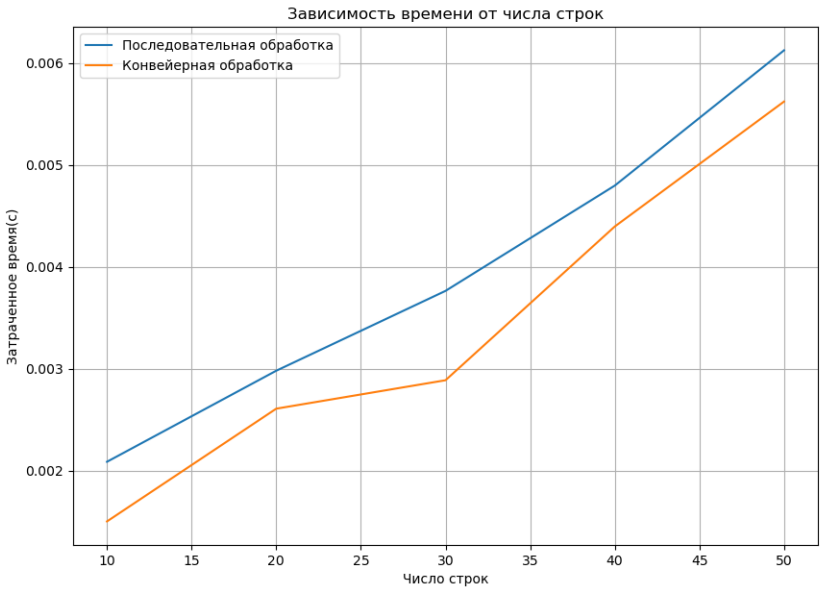

Рисунок 4.3 – Сравнение по времени в зависимости от числа строк

Кроме того, замеры времени проводились для строк-палиндромов, заполненных случайным образом, длиной от 10000 до 100000 символов с шагом в 10000 символов. Результаты измерения времени в зависимости от длины строк приведены в таблице 4.2 (в с).

На рисунке 4.4 приведены графические результаты сравнения временных характеристик для различной длины строк.

Таблица 4.2 – Результаты замеров времени в зависимости от порядка графа

Число строк	Конвейерная	Последовательная
10000	0.00353928	0.00423032
20000	0.00321933	0.00492197
30000	0.00478432	0.00982671
40000	0.0114914	0.0148205
50000	0.0132036	0.0195122
60000	0.0230261	0.0263445
70000	0.0223084	0.0308759
80000	0.0344795	0.0368916
90000	0.0372315	0.0488796
100000	0.0499187	0.0527271

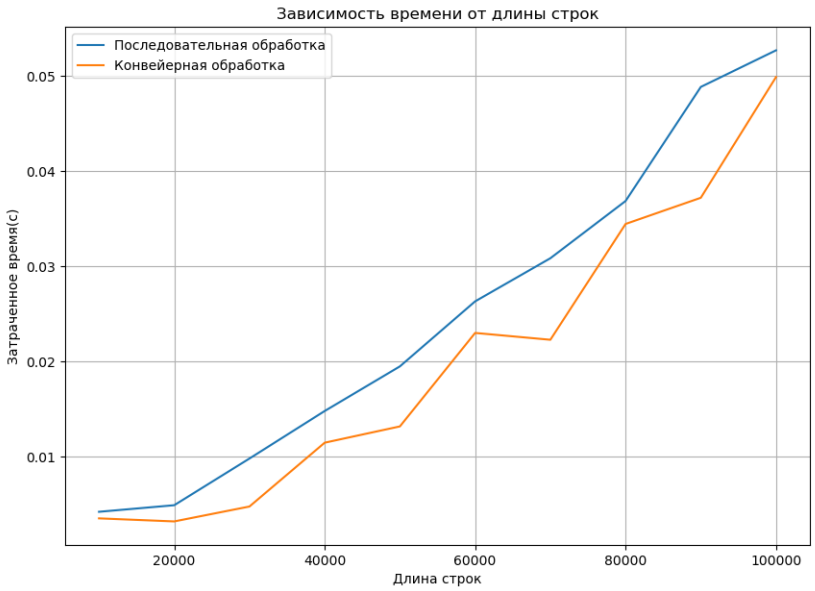

Рисунок 4.4 – Сравнение по времени в зависимости от длины строк

4.4 Вывод

В результате эксперимента было получено, что конвейерная обработка 30 строк в 1.3 раза быстрее последовательной обработки строк-палиндромов длиной 6000 символов. Тогда, для указанных данных небходимо использо-

вать конвейерную обработку данных.

Также в результате сравнительного анализа было установлено, что при увеличении длины строк конвейерная обработка быстрее последовательной: при длине, равной 20000 символам - в 1.5 раза, при длине, равной 30000 - в 2 раза. Можно сделать вывод, что конвейерную обработку следует применять при больших значениях длины строк.

Заключение

В результате исследования было получено, что на устройстве, использованном для тестирования, конвейерная обработка набора строк показывает лучшие результаты - быстрее последовательной обработки 30 строк в 1.3 раза для длины строк-палиндромов, равной 6000 символов. При этом конвейерную обработку следует использовать на больших значениях длины строк, так как для длины, равной 20000 символам, обработка происходит быстрее последовательной в 1.5 раз, а на длине, равной 30000 - в 2 раза.

Цель, поставленная перед началом работы, была достигнута. В ходе лабораторной работы были решены следующие задачи:

- были изучены последовательная и конвейерная обработка набора строк;
- были разработаны изученные алгоритмы;
- был проведен сравнительный анализ реализованных алгоритмов;
- был подготовлен отчет о выполненной лабораторной работе.

Список литературы

[1] Параллельная обработка данных [Электронный ресурс]. Режим доступа: https://parallel.ru/vvv/lec1.html (дата обращения: 25.11.2021).

[2] Работа со строками в C [Электронный ресурс]. Режим доступа: https://foxford.ru/wiki/informatika/rabota-so-strokami-v-s (дата обращения: 25.11.2021).

[3] Документация по языку C++ [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170 (дата обращения: 8.11.2021).

[4] Date and time utilities [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono (дата обращения: 8.11.2021).

[5] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 8.11.2021).

[6] Ubuntu 20.04 LTS (Focal Fossa) Beta [Электронный ресурс]. Режим доступа: http://old-releases.ubuntu.com/releases/20.04.1/ (дата обращения: 8.11.2021).

[7] Мобильный процессор AMD Ryzen™ 3 3200U с графикой Radeon™ Vega 3 [Электронный ресурс]. Режим доступа: https://www.amd.com/ru/products/apu/amd-ryzen-3-3200u (дата обращения: 8.11.2021).