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 №4 по курсу «Анализ алгоритмов»

Тема Параллельные вычисления на основе нативных потоков

Студент Ву Хай Данг

Группа ИУ7-52Б

Оценка (баллы) _____

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

Москва — 2024 г.

Содержание

Введение **3**

1 Аналитическая часть **4**

1.1 Последовательный алгоритм поиска подстроки в строке . . . 4

1.2 Параллельный алгоритм поиска подстроки в строке 4

2 Конструкторская часть **6**

2.1 Разработка алгоритмов 6

3 Технологическая часть **13**

3.1 Требования к программному обеспечению 13

3.2 Средства реализации 14

3.3 Сведения о модулях программы 14

3.4 Реализация алгоритмов 14

3.5 Функциональные тесты 16

4 Исследовательская часть **18**

4.1 Технические характеристики 18

4.2 Демонстрация работы программы 18

4.3 Время выполнения реализаций алгоритмов 19

4.4 Вывод 22

Заключение **23**

Список использованных источников **24**

Введение

В настоящее время компьютерные системы оперируют большими объемами данных. Над этими данными проводится большой объем различного рода вычислений. Для того, чтобы они выполнялись быстрее, было придумано параллельное программирование. Параллельное программирование — это подход к разработке программ, в котором задачи выполняются на нескольких процессорах или ядрах процессора [1].

Его суть заключается в том, чтобы относительно равномерно разделять нагрузку между потоками ядра. Каждое из ядер процессора может обрабатывать по одному потоку, поэтому когда количество потоков на ядро становится больше, происходит квантование времени. Это означает, что на каждый процесс выделяется фиксированная величина времени (квант), после чего в течение кванта обрабатывается следующий процесс. Таким образом создается видимость параллельности. Тем не менее, данная оптимизация может сильно ускорить вычисления [2].

Целью данной лабораторной работы является получение навыков параллельного программирования на базе алгоритма поиска подстроки в строке полным перебором. Для достижения поставленной цели требуется выполнить следующие задачи:

— описать последовательный и параллельный алгоритмы поиска подстроки в строке полным перебором;

— построить схемы данных алгоритмов;

— создать программное обеспечение, реализующее рассматриваемые алгоритмы;

— провести сравнительный анализ по времени для реализованного алгоритма;

— подготовить отчет о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будут представлены описания последовательного и параллельного вариантов алгоритма поиска подстроки в строке полным перебором.

1.1 Последовательный алгоритм поиска подстроки в строке

Данный алгоритм можно определить следующим образом [3]. Пусть задана строка S из N элементов и строка p из M элементов.

Описаны они так: $stringS[N], P[M]$;

Задача поиска подстроки P в строке S заключается в нахождении первого слева вхождения P в S, т.е. найти значение индекса i, начиная с которого

$S[i] = P[0], S[i+1] = P[1], \ldots, S[i+M-1] = P[M-1]$

1.2 Параллельный алгоритм поиска подстроки в строке

Идея алгоритма состоит в том, чтобы разделить строку на подстроку, соответствующие каждому потоку, а затем выполнить поиска для каждой подстроку. Найденные индексы сохраняются в массиве, поэтому может возникнуть ошибка, поскольку к массиву может обращаться несколько потоки. Для этого использовать мьютекс для организации монопольного доступа к массиву.

Основная проблема этой версии заключается в том, что если паттерн поступает в часть разделения данных или точку соединения, он не обнаруживается, поскольку данные обрабатываются на разных потоках. Для решения этой проблемы мы обрабатываем еще часть строки в каждой точке соединения.

Вывод

Были изучены последовательный и параллельный алгоритмы поиска подстроки в строке.

2 Конструкторская часть

В данном разделе будут представлены схемы последовательного и параллельного алгоритмов поиска подстроки в строке полным перебором.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема последовательного алгоритма поиска подстроки в строке полным перебором. Схема параллельного алгоритма поиска подстроки в строке полным перебором на рисунках 2.2–2.3. Схема алгоритма задачи одного потока представлена на рисунках 2.4–2.5.

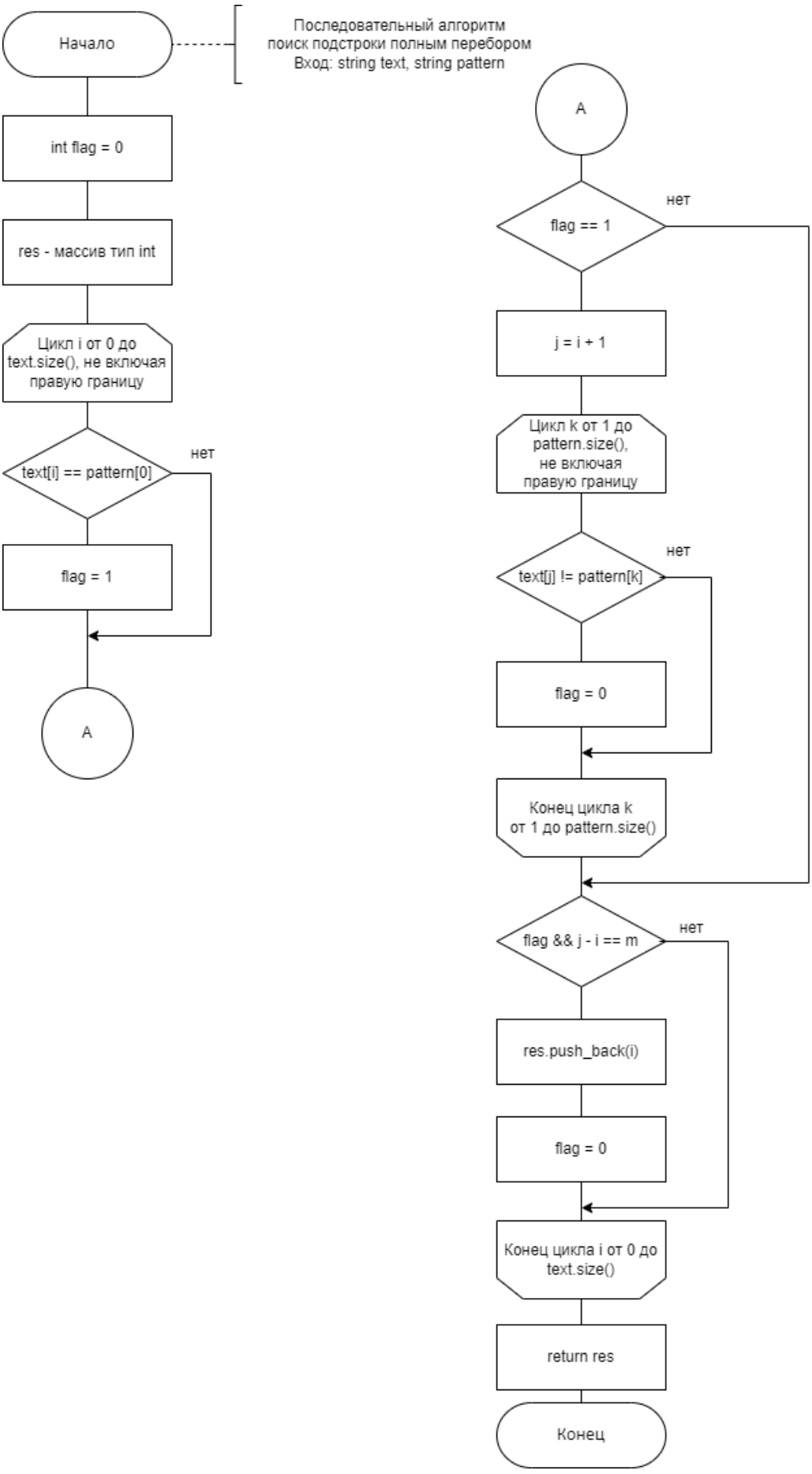

Рисунок 2.1 – Алгоритм последовательного поиска подстроки в строке полным перебором

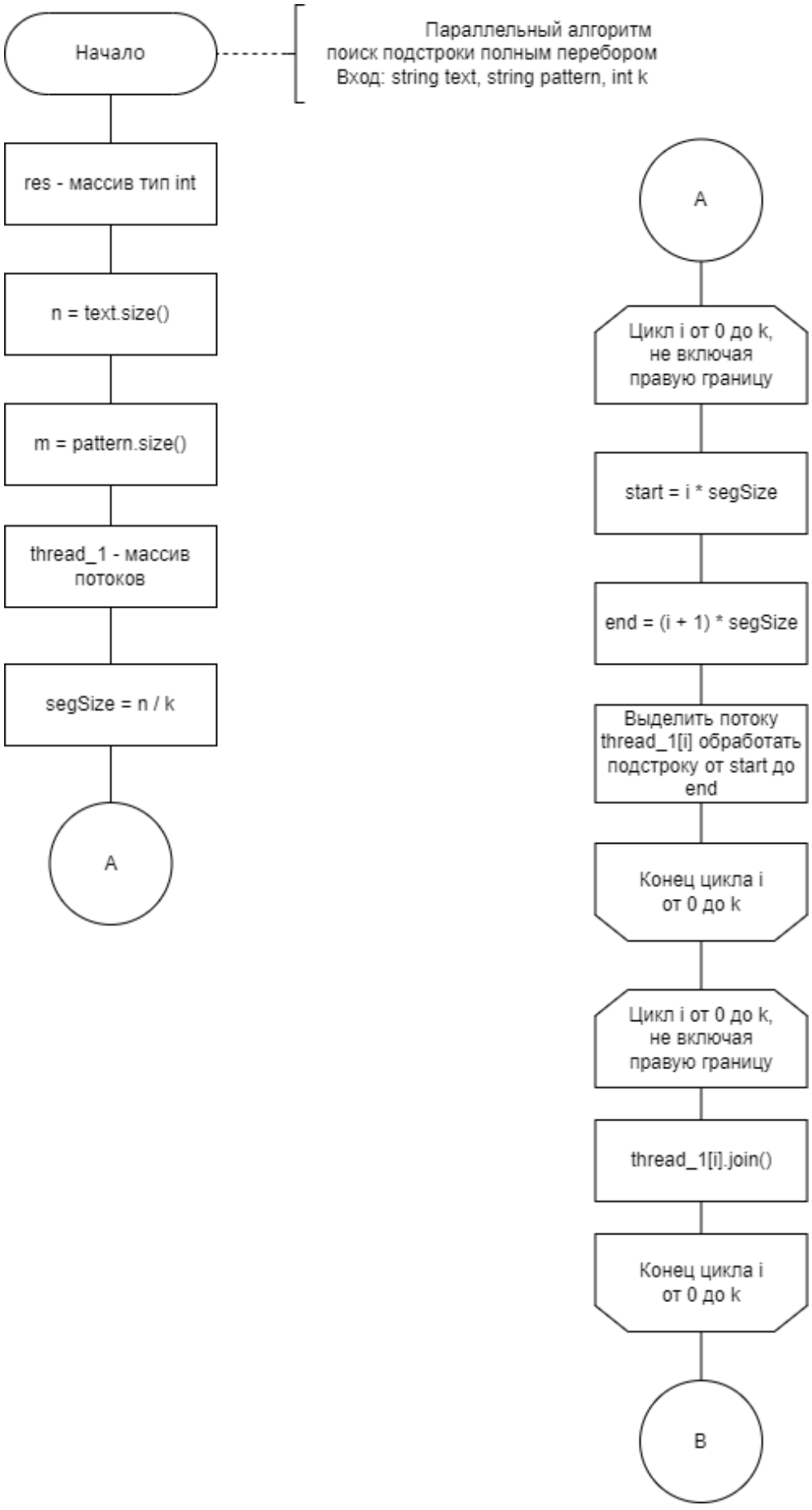

Рисунок 2.2 – Алгоритм параллельного поиска подстроки в строке полным перебором

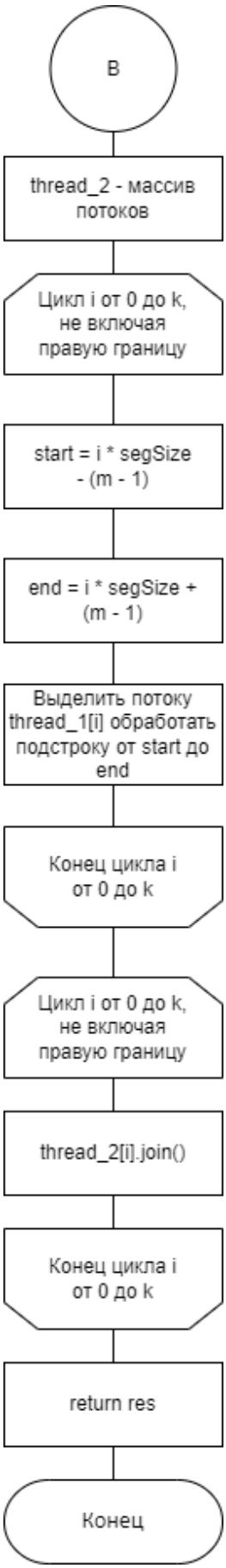

Рисунок 2.3 – Продолжение алгорита параллельного поиска подстроки в строке полным перебором

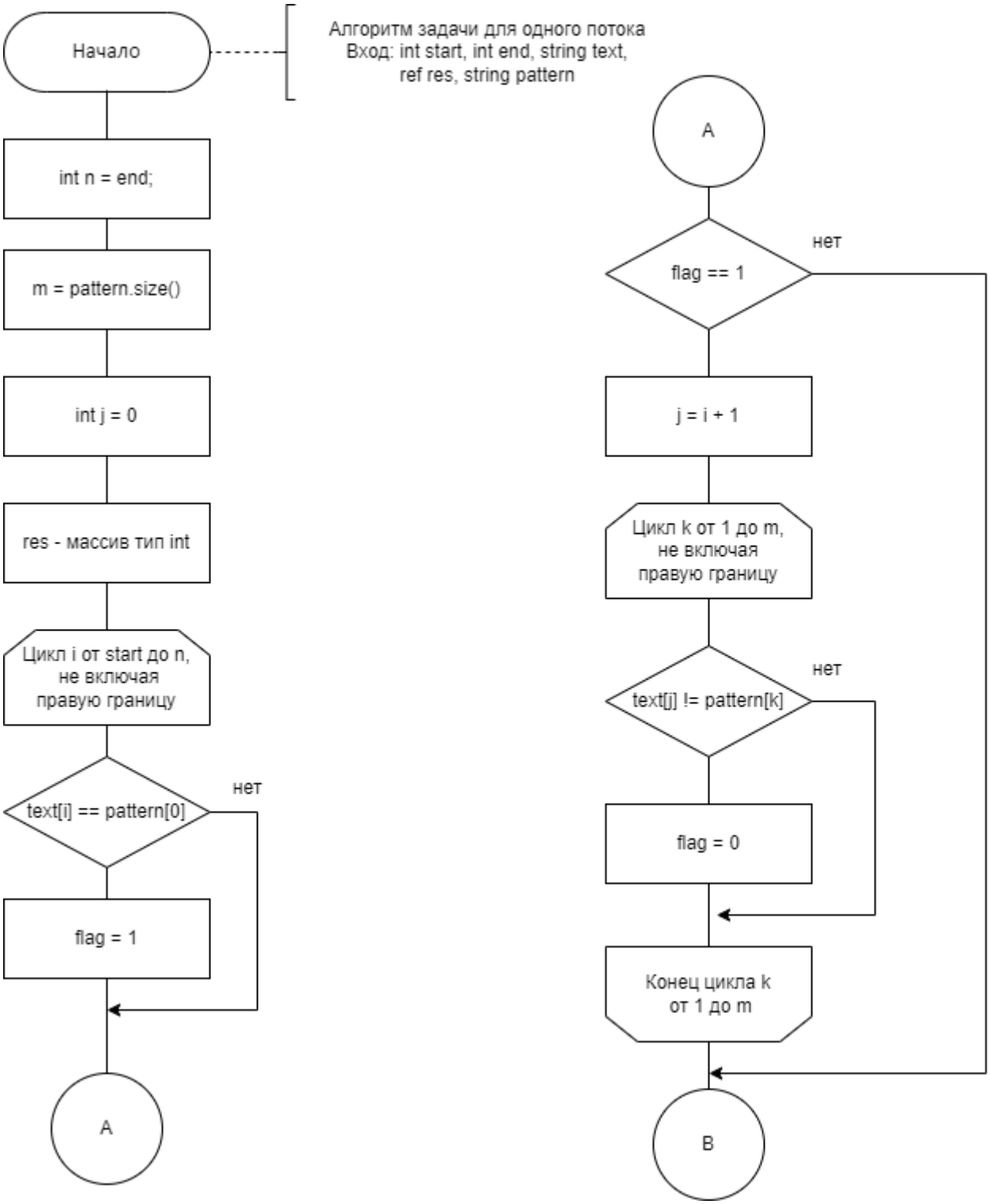

Рисунок 2.4 – Алгоритм задачи одного потока

Рисунок 2.5 – Продолжение алгоритма задачи одного потока

Вывод

Были представлены схемы последовательного и параллельного алгоритмов поиска подстроки в строке полным перебором.

3 Технологическая часть

В данном разделе будут указаны требования к программному обеспечению, средства реализации, будут представлены реализации алгоритмов, а также функциональные тесты.

3.1 Требования к программному обеспечению

К программе предъявляются следующие требования.

— Программа должна предоставлять 2 режима работы: режим поиска подстроки в строке и режим замера процессорного времени выполнения поиска подстроки в строке.

— В начале работы программы пользователю нужно ввести целое число — это выбор пункта меню.

К первому режиму работы программы предъявляются следующие требования:

— если пункт меню — число от 1 до 2, то искать подстроки в строке, для этого надо ввести две строки;

— программа должна вывести индексы в заданной строке.

Ко второму режиму работы программы предъявляются следующие требования:

— если пункт меню — число 3, то провести замеры времени поиска подстроки в строке каждого алгоритма;

— строки генерируются автоматически.

3.2 Средства реализации

Реализация данной лабораторной работы выполнялась при помощи языка программирования C++. Данный выбор обусловлен наличием у языка функции *clock*() измерения процессорного времени.

Визуализация графиков с помощью библиотеки *Matplotlib* [4].

3.3 Сведения о модулях программы

Программа состоит из следующих модулей:

— main.cpp — точка входа программы;

— algorithm.h — заголовочный файл, содержащий объявлении функций, реализуюших рассматриваемых алгоритмов;

— algorithm.cpp — файл, содержащий реализации этих функций;

— measure.h — заголовочный файл, содержащий объявлении функций замеров времени работы рассматриваемых алгоритмов;

— measure.cpp —algo файл, содержащий реализации функций замеров времени работы рассматриваемых алгоритмов.

3.4 Реализация алгоритмов

Реализации последовательного и параллельного алгоритмов приведены в листингах 3.1–3.3.

Листинг 3.1 – Последовательный алгоритм поиска подстроки в строке полным перебором

```
1 vector<int> algoS(string text, string pattern)
2 {
3     int n = text.size(), m = pattern.size();
4     vector<int> res;
5 
6     int flag = 0;
```

```
    int j = 0;
    for (int i = 0; i < n; i++)
    {
        if (text[i] == pattern[0])
            flag = 1;
        if (flag)
        {
            j = i + 1;
            for (int k = 1; flag && k < m && j < n; j++, k++)
                if (text[j] != pattern[k])
                {
                    flag = 0;
                }
        }
        if (flag && j - i == m)
            res.push_back(i);
        flag = 0;
    }
    return res;
```

Листинг 3.2 – Параллельный алгоритм поиска подстроки в строке полным перебором

```
vector<int> parallelS(string text, string pattern, int k)
{
    vector<int> res;
    int n = text.size(), m = pattern.size();

    vector<thread> threads_1;
    int segmentSize = n / k;
    int start, end;
    for (int i = 0; i < k; i++)
    {
        start = i * segmentSize;
        end = (i == k - 1) ? n : (i + 1) * segmentSize;
        threads_1.emplace_back(threadS, start, end, ref(pattern),
            ref(text), ref(res));
    }
    for (auto& th : threads_1)
        th.join();

    vector<thread> threads_2;
    for (int i = 1; i <= k - 1; i++)
    {
        start = i * segmentSize - (m - 1);
        end = i * segmentSize + (m - 1);
        threads_2.emplace_back(threadS, start, end, ref(pattern),
            ref(text), ref(res));
```

```
    }
    for (auto& th : threads_2)
        th.join();
    return res;
}
```

Листинг 3.3 – Задача одного потока

```
void threadS(int start, int end, string& pattern, string& text,
    vector<int>& res)
{
    int n = end, m = pattern.size();

    int flag = 0;
    int j = 0;
    for (int i = start; i < n; i++)
    {
        if (text[i] == pattern[0])
            flag = 1;
        if (flag)
        {
            j = i + 1;
            for (int k = 1; flag && k < m && j < n; j++, k++)
                if (text[j] != pattern[k])
                {
                    flag = 0;
                }
        }
        if (flag && j - i == m)
        {
            m_res.lock();
            res.push_back(i);
            m_res.unlock();
            flag = 0;
        }
    }
```

3.5 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для функции, реализующей алгоритм поиска подстроки в строке. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

	Входные данные		**Ожидаемый результат**
№	**Строка**	**Подстрока**	**Алгоритм полным перебором**
1			Сообщение об ошибке
2	anhyeuem	em	6
3	abcdabc	bc	1 5
4	anhyeuem	z	

Вывод

Были представлены реализаций двух версий алгоритма поиска подстроки в строке полным перебором — последовательного и параллельного. Также в данном разделе была приведены функциональные тесты.

4 Исследовательская часть

В данном разделе будут приведены демонстрации работы программы, и будет проведен сравнительный анализ реализованного алгоритма поиска подстроки в строке полным перебором по времени.

4.1 Технические характеристики

Ниже представлены характеристики компьютера, на котором проводились замеры времени работы реализации алгоритмов:

— операционная система Windows 10 Домашняя 21H2;

— оперативная память 12 Гб;

— процессор Intel(R) Core(TM) i7-9750H CPU @ 2.6ГГц.

4.2 Демонстрация работы программы

На рисунках 4.1 и 4.2 представлен результат работы программы. В каждом примере пользователем введены строка и подстрока и получены все индексы вхождения.

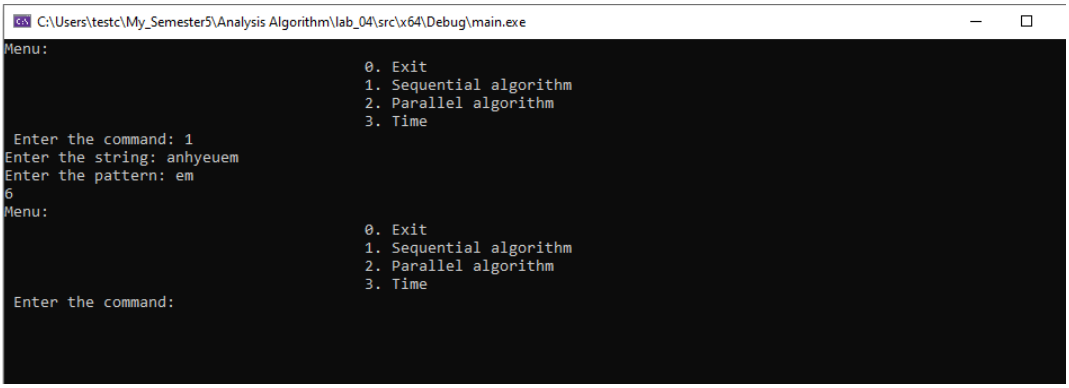

Рисунок 4.1 – Демонстрация работы программы при последовательном алгоритме

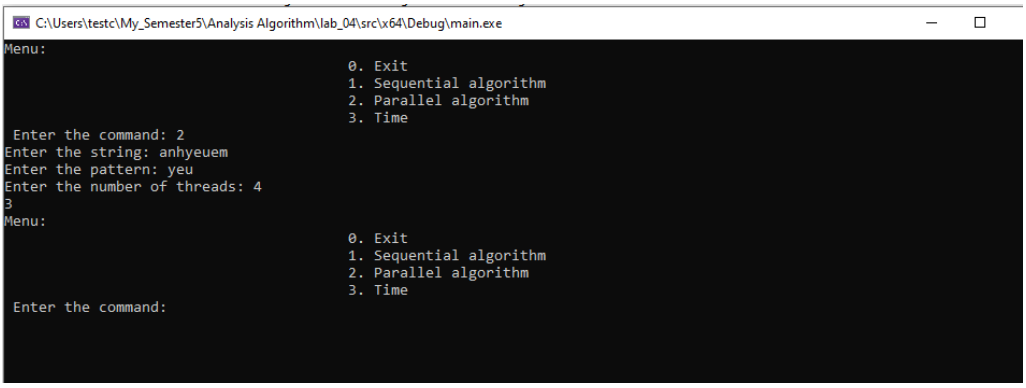

Рисунок 4.2 – Демонстрация работы программы при параллельном алгоритме

4.3 Время выполнения реализаций алгоритмов

Результаты замеров времени работы реализаций алгоритмов поиска подстроки в строке преведены в таблице 4.1. Для параллельной реализации алгоритма выбрано количество потоков, равное 12.

Таблица 4.1 – Результаты замеров времени

Размер строки	**Размер подстроки**	**Без многопоточности**	**с 12 потоками**
10000	5	3.7	2.8
20000	5	7.3	3.7
30000	5	10.8	4.9
40000	5	14.8	5.8
50000	5	19.9	6.7
60000	5	22.4	7.1

По таблице 4.1 был построен график, который иллюстрирует зависимость времени, затраченного реализациями алгоритмов поиска подстроки в строке полным перебором, от размера строки — рис. 4.3.

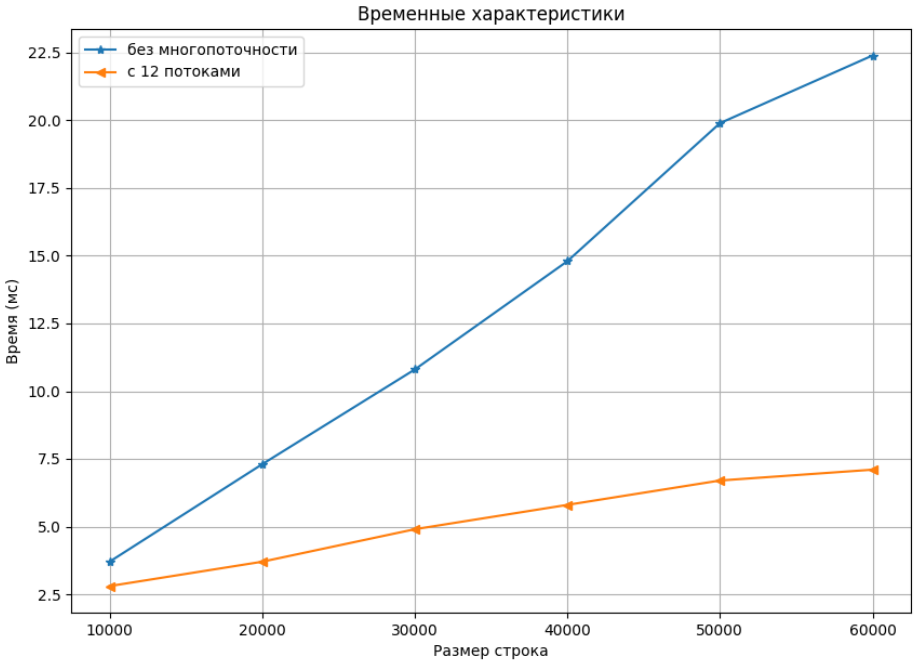

Рисунок 4.3 – Сравнение времени работы алгоритма без распараллеливания и с 12 вспомогательными потоками при разных размерах строки

В таблице 4.2 приведены результаты замеров по времени параллельного алгоритма поиска подстроки в строке полным перебором при разном количество потоков.

Таблица 4.2 – Результаты замеров времени

Количество потоков	**Время выполнения**
1	183.3
2	95.8
4	54.2
8	44
12	37.9
16	49.4

По таблице 4.2 был построен график, который иллюстрирует зависимость времени, затраченного реализацией параллельного алгоритма поиска подстроки в строке полным перебором, от количества потоков — рис. 4.4.

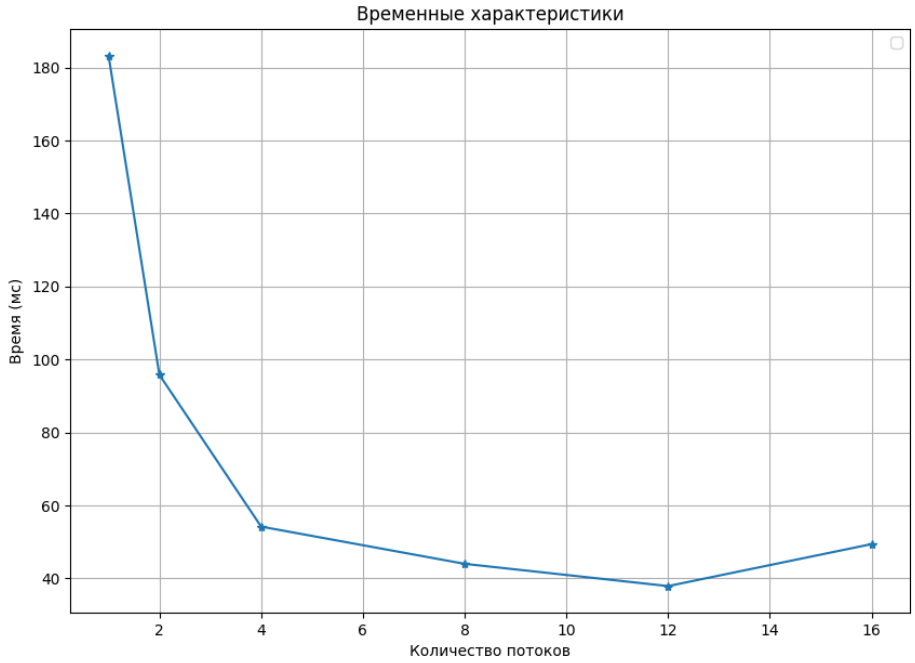

Рисунок 4.4 – Сравнение времени работы алгоритма с распараллеливанием на различное количество потоков при размере строки 5000000

Исходя из этих данных можно понять, параллельный алгоритм работает быстрее всего с 12 потоками.

4.4 Вывод

По графикам видно, что параллельный алгоритм с использованием 12 потоков, работает быстрее, чем последовательный алгоритм поиска подстроки в строке полным перебором. Сравнивая параллельный алгоритм с разным количеством потоков видно, что количество потоков 12 обеспечивает наилучшую производительность.

Заключение

По графикам видно, что параллельный алгоритм с использованием 12 потоков, работает быстрее, чем последовательный алгоритм поиска подстроки в строке полным перебором. Сравнивая параллельный алгоритм с разным количеством потоков видно, что количество потоков 12 обеспечивает наилучшую производительность.

Цель лабораторной работы была достигнута, в ходе выполнения лабораторной работы были решены все задачи:

— описаны последовательный и параллельный алгоритмы поиска подстроки в строке полным перебором;

— построены схемы данных алгоритмов;

— создано программное обеспечение, реализующее рассматриваемые алгоритмы;

— проведен сравнительный анализ по времени для реализованного алгоритма;

— подготовлен отчет о выполненной лабораторной работе.

Список использованных источников

1. Параллельное программирование и параллельные вычисления [Электронный ресурс]. Режим доступа: https://ya.zerocoder.ru/pgt-parallelnoe-programmirovanie-i-parallelnye-vychisleniya/

2. ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ [Электронный ресурс]. Режим доступа: https://kpfu.ru/staff_files/F1222332197/ParVychGafGal.pdf

3. Поиск подстроки в строке [Электронный ресурс]. Режим доступа: https://kpfu.ru/staff_files/F1222332197/ParVychGafGal.pdf

4. Matplotlib documentation [Электронный ресурс]. Режим доступа: https://matplotlib.org/stable/index.html