



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 Организация асинхронного взаимодействия потоков вычисления на примере
конвейерных вычислений

Студент Жаворонкова А. А.

Группа ИУ7-56Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Содержание

ВВЕДЕНИЕ	3
1 Аналитическая часть	4
1.1 Конвейерная обработка	4
1.2 Алгоритм Кнута — Морриса — Пратта	4
1.3 Алгоритм Бойера — Мура	5
1.4 Вывод	6
2 Конструкторская часть	8
2.1 Разработка алгоритмов	8
2.2 Вывод	12
3 Технологическая часть	13
3.1 Средства реализации	13
3.2 Описание используемых типов данных	13
3.3 Сведения о модулях программы	13
3.4 Реализация алгоритмов	14
3.5 Вывод	18
4 Исследовательская часть	19
4.1 Технические характеристики	19
4.2 Демонстрация работы программы	19
4.3 Время выполнения алгоритмов	20
4.4 Вывод	21
ЗАКЛЮЧЕНИЕ	23
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	24

ВВЕДЕНИЕ

Целью данной работы является изучение асинхронного взаимодействия потоков вычисления на примере конвейерных вычислений. Для достижения поставленной цели необходимо выполнить следующие задачи:

- описать конвейерную обработку;
- описать алгоритмы Кнута — Морриса — Пратта и Бойера — Мура;
- реализовать конвейерную обработку с указанными алгоритмами;
- провести тестирование по методу черного ящика для реализации конвейера;
- провести сравнительный анализ зависимости времени обработки от количества заявок;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

1 Аналитическая часть

В данном разделе будет рассмотрена конвейерная обработка, а также алгоритмы Кнута — Морриса — Пратта и Бойера — Мура.

1.1 Конвейерная обработка

Конвейер — это механизм, предназначенный для распараллеливания выполнения команд программы между блоками процессора. Он позволяет загрузить блоки процессора при выполнении команд оптимально, без простоев [4]. Заявка — структура с временем и флагом отбоя.

Генератор заранее генерирует N заявок и помещает их в первую очередь. Из первой очереди обслуживающее устройство 1 берет заявку и выполняет первую стадию обработки, после чего помещает ее во вторую очередь. Из второй очереди берет заявку обслуживающее устройство 2. И так далее. В результате последнее обслуживающее устройство помещает заявку в пул обработанных заявок. Когда все заявки будут обработаны (конвейер завершит свою работу), по пулу будет сформирована статистика и выведена в лог-файл. При этом каждый рабочий поток существует, пока не получит специальную заявку с установленным флагом отбоя.

Поскольку очереди заявок являются разделяемой памятью, для обеспечения монопольного доступа к ним необходимо использовать мьютекс. Мьютексы представляют собой объекты ядра, используемые для синхронизации, регулирующие доступ к единственному ресурсу [6].

В данной работе обслуживающее устройство 1 выполняет поиск подстроки в файле по алгоритму Кнута — Морриса — Пратта. Обслуживающее устройство 2 — по алгоритму Бойера — Мура. Обслуживающее устройство 3 выполняет запись результатов поиска в файл.

1.2 Алгоритм Кнута — Морриса — Пратта

Алгоритм Кнута — Морриса — Пратта используется для поиска специальных подстрок с повторами префиксов [3]. Его основная идея — построение

автомата для определения величин смещения. При смещении подстроки после n успешных сравнений и 1 неуспешного считается, что одно сравнение префикса в активе и его не нужно проверять.

Рассмотрим пример. Пусть искомая подстрока — *ababcb*. В построенном автомате состояния маркируются проверяемым в нем символом, дугу — успехом (*s*) или неудачей (*f*). Полученный автомат представлен на рисунке 1.1.

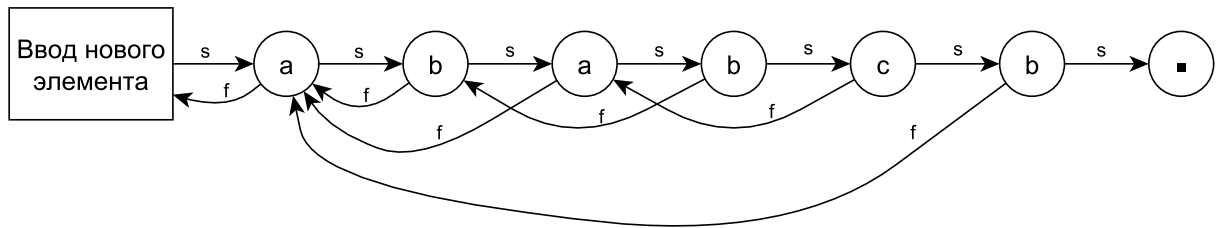


Рисунок 1.1 – Автомат для определения величин смещения

В результате будет получен массив сдвигов, который рассчитывается один раз и используется повторно. Для примера выше таким массивом будет: $[1, 1, 2, 2, 2, 5]$

1.3 Алгоритм Бойера — Мура

Преимущество этого алгоритма в том, что необходимо сделать некоторые предварительные вычисления над подстрокой, чтобы сравнение подстроки с исходной строкой осуществлять не во всех позициях — часть проверок пропускаются как заведомо не дающие результата [5].

Первоначально строится таблица смещений для искомой подстроки. Далее идет совмещение начала строки и подстроки и начинается проверка с последнего символа подстроки. Если последний символ подстроки и соответствующий ему при наложении символ строки не совпадают, подстрока сдвигается относительно строки на величину, полученную из таблицы смещений, и снова проводится сравнение, начиная с последнего символа подстроки. Если же символы совпадают, производится сравнение предпоследнего символа подстроки и так далее. Если все символы подстроки совпали с наложенными символами строки, значит, найдена подстрока и поиск окончен. Если же какой-то (не последний) символ подстроки не совпадает с соответствующим

символом строки, далее производим сдвиг подстроки на один символ вправо и снова выполняется проверка с последнего символа. Весь алгоритм выполняется до тех пор, пока либо не будет найдено вхождение искомой подстроки, либо не будет достигнут конец строки. Величина сдвига в случае несовпадения последнего символа вычисляется, исходя из следующего: сдвиг подстроки должен быть минимальным, таким, чтобы не пропустить вхождение подстроки в строке. Если данный символ строки встречается в подстроке, то подстрока смещается таким образом, чтобы символ строки совпал с самым правым вхождением этого символа в подстроке. Если же подстрока вообще не содержит этого символа, то подстрока сдвигается на величину, равную ее длине, так что первый символ подстроки накладывается на следующий за проверявшимся символом строки.

Величина смещения для каждого символа подстроки зависит только от порядка символов в подстроке, поэтому смещения удобно вычислить заранее и хранить в виде одномерного массива, где каждому символу алфавита соответствует смещение относительно последнего символа подстроки.

1.4 Вывод

В данном разделе была теоретически разобрана конвейерная обработка, а также алгоритмы Кнута — Морриса — Пратта и Бойера — Мура.

К разрабатываемой программе предъявляются следующие требования:

1. Программа должна предоставлять функциональность конвейерной обработки;
2. Реализуемое ПО будет работать в двух режимах — пользовательском, в котором можно выбрать алгоритм и вывести для него результат, а также экспериментальном режиме, в котором можно произвести сравнение времени обработки в зависимости от количества заявок на различных входных данных;
3. В первом режиме в качестве входных данных в программу будет подаваться текстовые файлы, также реализовано меню для вызова конвейерной обработки и замеров времени;

4. Во втором режиме будет происходить измерение процессорного времени, будут построены зависимости времени обработки всех заявок от их количества. Заявки будут сгенерированы автоматически для заданного размера искомых подстрок.

2 Конструкторская часть

В этом разделе будут представлены схемы алгоритмов Кнута — Морриса — Пратта и Бойера — Мура, а также конвейерной обработки.

2.1 Разработка алгоритмов

На рисунках 2.1 — 2.4 представлены схемы алгоритмов Кнута — Морриса — Пратта и Бойера — Мура, а также конвейерной обработки.

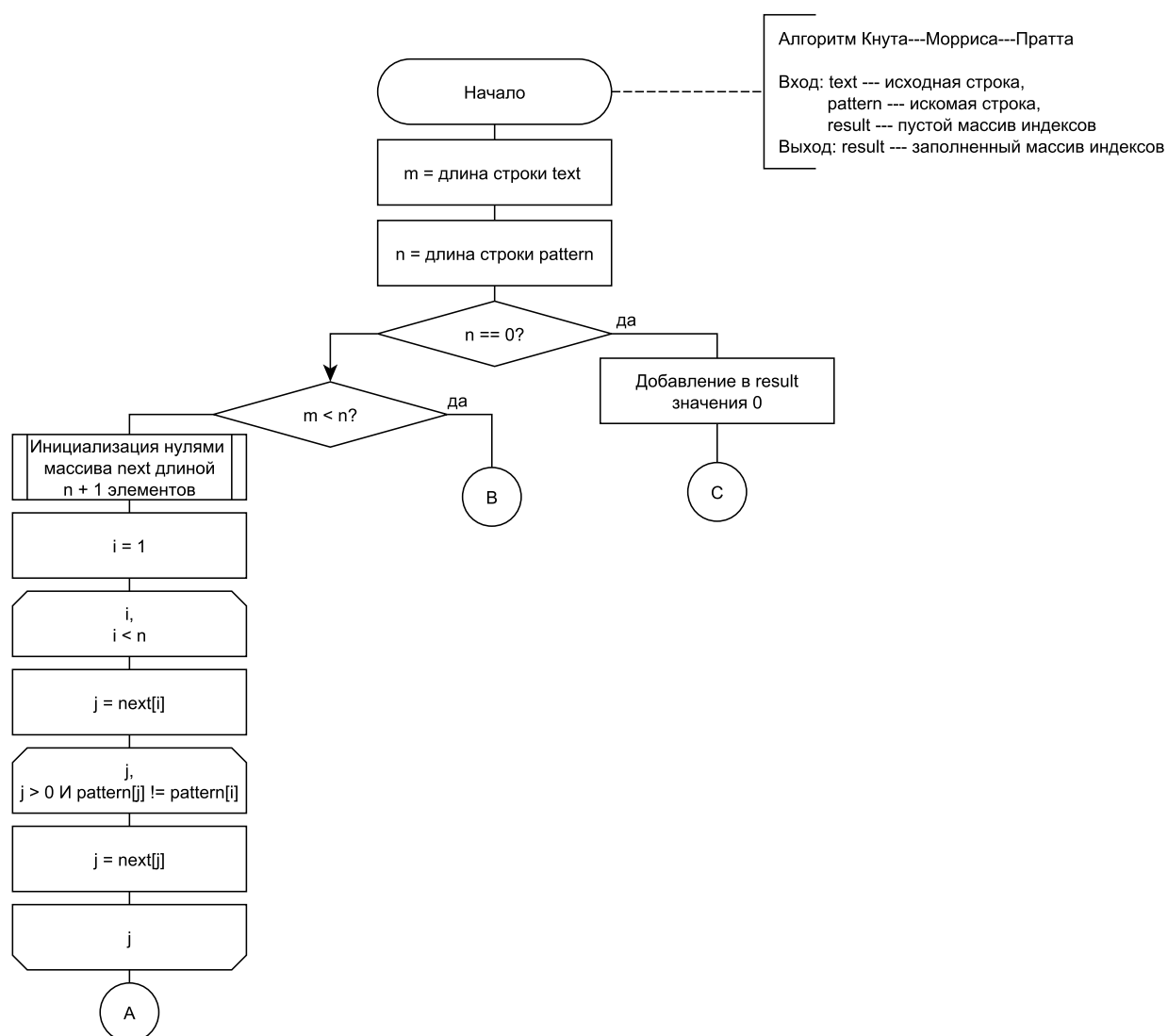


Рисунок 2.1 – Схема алгоритма Кнута — Морриса — Пратта (часть 1)

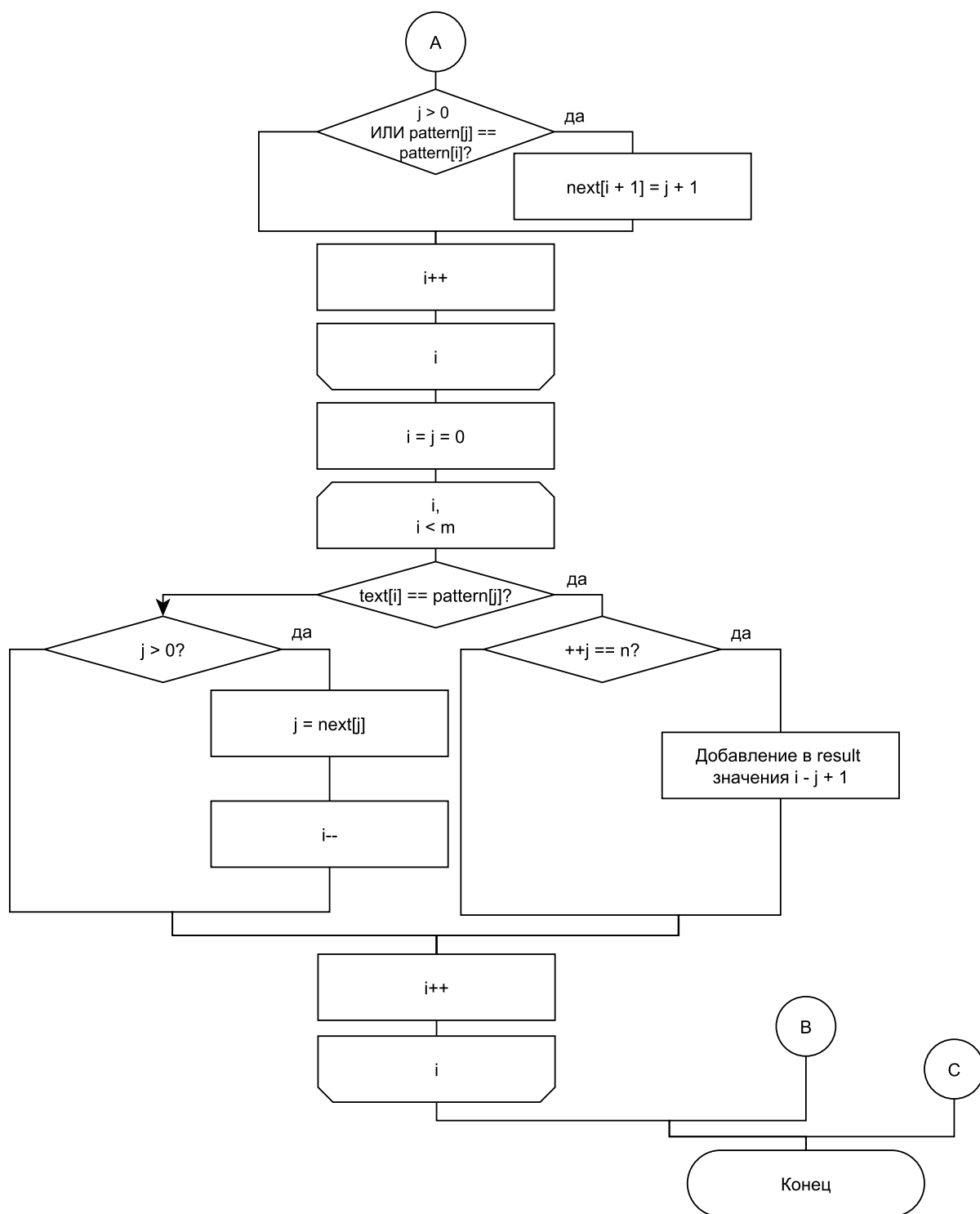


Рисунок 2.2 – Схема алгоритма Кнута — Морриса — Пратта (часть 2)

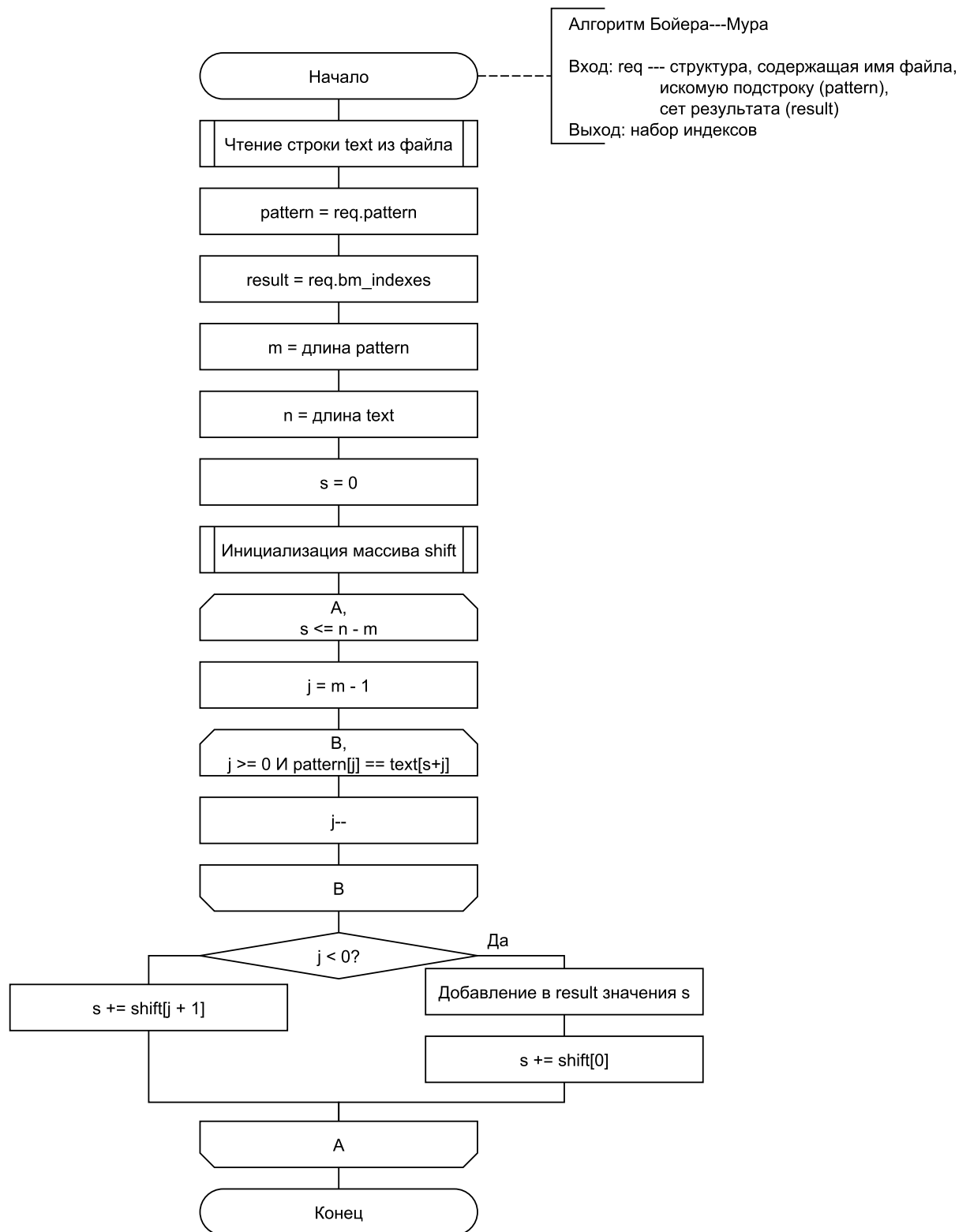


Рисунок 2.3 – Схема алгоритма Бойера — Мура

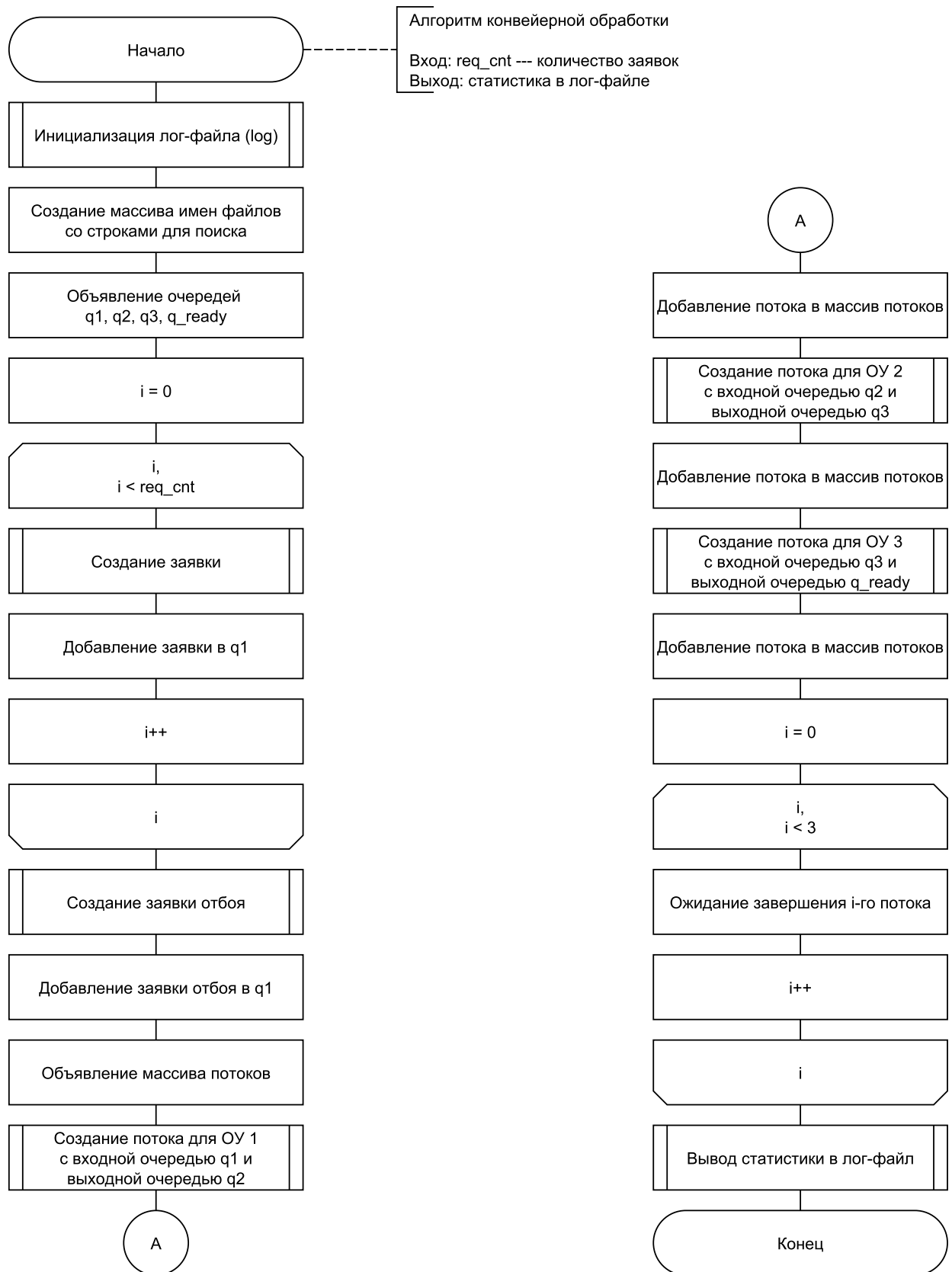


Рисунок 2.4 – Схема алгоритма конвейерной обработки

2.2 Вывод

В данном разделе были представлены схемы алгоритмов, рассматриваемых в лабораторной работе.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги реализаций алгоритмов Кнута — Морриса — Пратта и Бойера — Мура, а также конвейерной обработки.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *C++* [2]. В текущей лабораторной работе требуется замерить процессорное время для выполняемой программы, а также реализовать принципы многопоточного алгоритма. Все эти инструменты присутствуют в выбранном языке программирования. Время замерено с помощью функции *std::chrono::system_clock::now()* из библиотеки *chrono* [1].

3.2 Описание используемых типов данных

При реализации будут использованы следующие типы и структуры данных:

- *std::set* — для результирующего набора найденных индексов;
- *std::string* — для исходной строки и искомой подстроки;
- *request_t* — структура заявки, содержащая имя файла для поиска, искомую подстроку, флаг завершения;
- *std::queue<request_t>* — для очередей заявок;
- *std::vector<std::thread>* — для потоков.

3.3 Сведения о модулях программы

Программа состоит из двух модулей:

- *main.cpp* — файл, содержащий весь служебный код;
- *conveyor.cpp* — файл, содержащий конвейерную обработку.

3.4 Реализация алгоритмов

В листингах 3.1 – 3.7 представлены реализации алгоритмов Кнута — Морриса — Пратта и Бойера — Мура, а также конвейерной обработки.

Листинг 3.1 – Реализация алгоритма Кнута — Морриса — Пратта (начало)

```
1 void kmp(request_t &req)
2 {
3     string text = read_from_file(req);
4     string pattern = req.pattern;
5     set<int>& result = req.kmp_indexes;
6
7     int m = text.length();
8     int n = pattern.length();
9     if (n == 0)
10    {
11        result.insert(0);
12        return;
13    }
14    if (m < n)
15        return;
16
17    // next[i] сохраняет индекс следующего лучшего частичного совпад
18    ения
19    int next[n + 1];
20    for (int i = 0; i < n + 1; i++) {
21        next[i] = 0;
22    }
23
24    for (int i = 1; i < n; i++)
25    {
26        int j = next[i];
27        while (j > 0 && pattern[j] != pattern[i]) {
28            j = next[j];
29        }
30        if (j > 0 || pattern[j] == pattern[i]) {
```

Листинг 3.2 – Реализация алгоритма Кнута — Морриса — Пратта
(окончание)

```
1      if (j > 0 || pattern[j] == pattern[i]) {
2          next[i + 1] = j + 1;
3      }
4  }
5
6  for (int i = 0, j = 0; i < m; i++)
7  {
8      if (text[i] == pattern[j])
9      {
10         if (++j == n) {
11             result.insert(i - j + 1);
12         }
13     }
14     else if (j > 0)
15     {
16         j = next[j];
17         i--;    // так как 'i' будет увеличен на следующей итера
                  ции
18     }
19 }
20 }
```

Листинг 3.3 – Реализация алгоритма Бойера — Мура (начало)

```
1 void preprocess_strong_suffix(int *shift, int *bpos, string pat,
   int m)
2 {
3     int i = m, j = m + 1;
4     bpos[i] = j;
5
6     while(i > 0)
7     {
8         while(j <= m && pat[i - 1] != pat[j - 1])
9         {
10             if (shift[j] == 0)
11                 shift[j] = j - i;
12
13             j = bpos[j];
14         }
15         i--; j--;
```

Листинг 3.4 – Реализация алгоритма Бойера — Мура (продолжение)

```

1      i--; j--;
2      bpos[i] = j;
3  }
4  }
5
6  void preprocess_case2(int *shift, int *bpos, string pat, int m)
7  {
8      int i, j;
9      j = bpos[0];
10     for(i = 0; i <= m; i++)
11     {
12         if(shift[i] == 0)
13             shift[i] = j;
14
15         if (i == j)
16             j = bpos[j];
17     }
18 }
19
20 void bm(request_t &req)
21 {
22     string text = read_from_file(req);
23     string pattern = req.pattern;
24     set<int>& result = req.bm_indexes;
25
26     int m = pattern.length();
27     int n = text.length();
28
29     int s = 0, j;
30     int bpos[m + 1], shift[m + 1];
31
32     for(int i = 0; i < m + 1; i++) shift[i] = 0;
33
34     preprocess_strong_suffix(shift, bpos, pattern, m);
35     preprocess_case2(shift, bpos, pattern, m);
36
37     while(s <= n - m)
38     {
39         j = m - 1;
40         while(j >= 0 && pattern[j] == text[s+j])

```


Листинг 3.5 – Реализация алгоритма Бойера — Мура (окончание)

```

1      while(j >= 0 && pattern[j] == text[s+j])
2          j--;
3      if (j < 0)
4      {
5          result.insert(s);
6          s += shift[0];
7      }
8      else
9          s += shift[j + 1];
10 }
11 }
```

Листинг 3.6 – Реализация алгоритма конвейерной обработки (начало)

```

1 void conveyor_start(int req_cnt, int str_len)
2 {
3     srand(time(NULL));
4     std::ofstream log("log.txt");
5     if (!log.is_open()) {
6         cout << "Не удалось открыть log файл" << endl;
7         return;
8     }
9     log << std::right << std::setw(8) << "id_req";
10    log << std::right << std::setw(20) << "time";
11    log << std::right << std::setw(11) << "stage_num" << endl;
12
13    std::vector<string> files = {"text1.c", "text2.c", "text3.c"};
14    // create req_cnt requests
15    std::queue<request_t> q1, q2, q3, q_ready;
16    for (int i = 0; i < req_cnt; ++i) {
17        request_t req = {.id = i + 1, .filename = files[rand() %
18            files.size()], .pattern = generate_str(str_len)};
19        q1.push(req);
20    }
21    request_t req = {.id = -1, .flag_end = true};
22    q1.push(req);
23
24    // create 3 threads
25    std::vector<std::thread> th_vect;
26    std::thread t1(stage1, std::ref(q1), std::ref(q2),
27        std::ref(log));
```

Листинг 3.7 – Реализация алгоритма конвейерной обработки (окончание)

```
1      std::thread t1(stage1, std::ref(q1), std::ref(q2),  
2          std::ref(log));  
3  
4      std::thread t2(stage2, std::ref(q2), std::ref(q3),  
5          std::ref(log));  
6  
7      std::thread t3(stage3, std::ref(q3), std::ref(q_ready),  
8          std::ref(log));  
9  
10     for (int i = 0; i < 3; ++i)  
11         th_vect[i].join();  
12  
13     log_statistic(q_ready, log);  
14 }
```

3.5 Вывод

Были представлены реализации алгоритмов Кнута — Морриса — Пратта и Бойера — Мура, а также конвейерной обработки, которые были описаны в предыдущем разделе. Также в данном разделе была приведена информация о выбранных средствах для разработки алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, а также проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

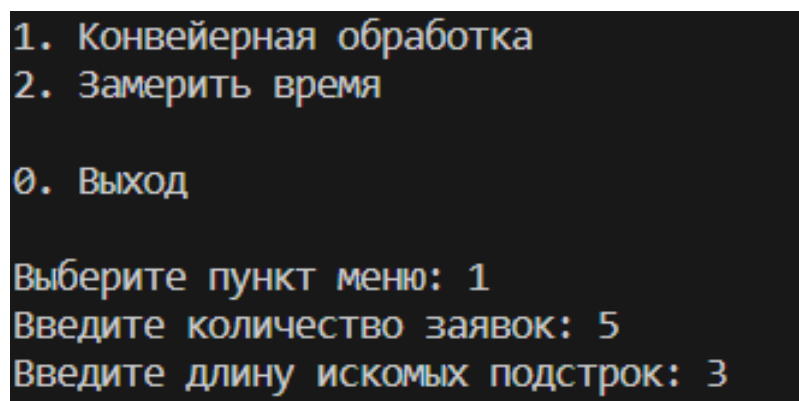
Технические характеристики устройства, на котором выполнялось тестирование представлены далее:

- операционная система: Windows 11, x64;
- оперативная память: 8 Гб;
- процессор: AMD Ryzen 5 5500U с видеокартой Radeon Graphics 2.10 ГГц.

Во время замеров времени ноутбук был нагружен только встроенными приложениями окружения.

4.2 Демонстрация работы программы

На рисунках 4.1 – 4.2 представлен результат работы программы.

A screenshot of a console application interface. It displays a menu with three options: '1. Конвейерная обработка', '2. Замерить время', and '0. Выход'. Below the menu, the user has entered '1' for the menu item, '5' for the number of requests, and '3' for the length of the substrings to search for. The text is displayed in a monospaced font with a light blue color on a black background.

```
1. Конвейерная обработка
2. Замерить время

0. Выход

Выберите пункт меню: 1
Введите количество заявок: 5
Введите длину искомых подстрок: 3
```

Рисунок 4.1 – Пример работы программы: вывод в консоль

id	req	time	stage	num
	1	2.89624	stage	1
	1	3.11931	stage	2
	2	3.65997	stage	1
	2	4.84712	stage	2
	3	5.40588	stage	1
	1	8.03533	stage	3
	3	5.07492	stage	2
	4	5.13913	stage	1
	2	9.08157	stage	3
	4	4.77629	stage	2
	5	5.62279	stage	1
	5	6.80242	stage	2
	3	9.63268	stage	3
	4	6.96965	stage	3
	5	5.86287	stage	3
Время обработки:				
		t_min	t_med	t_avg
		13	19	13
				t_max
				19

Рисунок 4.2 – Пример работы программы: вывод в лог-файл

4.3 Время выполнения алгоритмов

Как было сказано выше, используется функция замера процессорного времени `std::chrono::system_clock::now()`. Функция возвращает пользовательское процессорное время типа `float`. Использовать функцию приходится дважды, затем из конечного времени нужно вычесть начальное, чтобы получить результат.

Замеры проводились для количества заявок от 100 до 1000 с шагом 100. Результаты замеров приведены в таблице 4.1 (время в мс).

Таблица 4.1 – Результаты замеров времени

Количество заявок	Время обработки всех заявок, мс
100	67.869
200	121.505
300	179.139
400	240.659
500	300.084
600	357.260
700	418.859
800	483.653
900	540.842
1000	585.003

На рисунке 4.3 приведена визуализация результатов замеров.

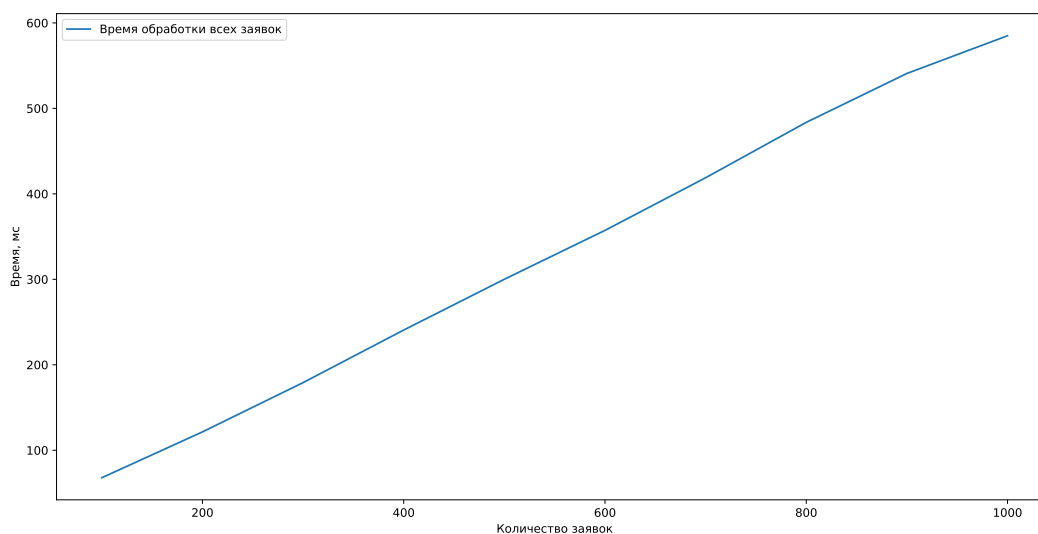


Рисунок 4.3 – Визуализация результатов замеров

4.4 Вывод

Как видно из графика 4.3, время обработки растет линейно в зависимости от количества заявок. Так, например, при увеличении количества заявок

в 10 раз, время увеличивается в 8,62 раза.

ЗАКЛЮЧЕНИЕ

В результате было получено, что время обработки линейно зависит от количества заявок.

Цель, которая была поставлена в начале лабораторной работы была достигнута: изучено асинхронное взаимодействие потоков вычисления на примере конвейерных вычислений. В ходе выполнения были решены все задачи:

- описана конвейерную обработку;
- описаны алгоритмы Кнута — Морриса — Пратта и Бойера — Мура;
- реализована конвейерную обработку с указанными алгоритмами;
- проведено тестирование по методу черного ящика для реализации конвейера;
- проведен сравнительный анализ зависимости времени обработки от количества заявок;
- описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. C++ chrono library. — URL: <https://cplusplus.com/reference/chrono/?kw=chrono> (дата обр. 05.12.2023).
2. C++ documnetation. — URL: <https://cplusplus.com/> (дата обр. 05.12.2023).
3. Математические основы алгоритмов. Строковые алгоритмы. Поиск в строке: алгоритм Кнута–Морриса–Пратта, его реализация на конечном автомате. — URL: https://users.math-cs.spbu.ru/~okhotin/teaching/algorithms1_2022/okhotin_algorithms1_2022_16.pdf (дата обр. 05.12.2023).
4. НОУ ИНТУИТ | Архитектура ЭВМ. Лекция 10: Конвейеризация. — URL: <https://intuit.ru/studies/courses/13849/1246/lecture/32770> (дата обр. 05.12.2023).
5. НОУ ИНТУИТ | Структуры и алгоритмы компьютерной обработки данных. Лекция 40: Алгоритмы поиска в тексте. — URL: <https://intuit.ru/studies/courses/648/504/lecture/11468?page=2> (дата обр. 05.12.2023).
6. НОУ ИНТУИТ. Лекция. Синхронизация потоков. — URL: https://new2.intuit.ru/studies/professional_skill_improvements/1717/courses/217/lecture/5599?page=2&ysclid=lpwnduox8g513624798 (дата обр. 05.12.2023).