



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 рубежному контролю №1 по курсу «Анализ алгоритмов»

Тема Алгоритм Кнута — Морриса — Пратта поиска подстроки в строке

Дополнение к лабораторной работе №4

Студент Жаворонкова А. А.

Группа ИУ7-56Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение	3
1 Аналитическая часть	4
1.1 Алгоритм Кнута — Морриса — Пратта	4
1.2 Вывод	4
2 Конструкторская часть	6
2.1 Разработка алгоритмов	6
2.2 Классы эквивалентности тестирования	8
2.3 Модель вычислений	8
2.4 Трудоемкость алгоритма	9
2.5 Вывод	11
3 Технологическая часть	12
3.1 Средства реализации	12
3.2 Описание используемых типов данных	12
3.3 Сведения о модулях программы	12
3.4 Реализация алгоритмов	12
3.5 Функциональные тесты	14
3.6 Вывод	14
4 Исследовательская часть	15
4.1 Технические характеристики	15
4.2 Демонстрация работы программы	15
4.3 Вывод	16
Заключение	17
Список используемых источников	18

Введение

Целью данной работы является применение алгоритма Кнута — Морриса — Пратта для поиска подстрок в бинарных строках. Для достижения поставленной цели необходимо выполнить следующие задачи:

- описать алгоритм Кнута — Морриса — Пратта;
- реализовать указанный алгоритм;
- рассчитать трудоемкость в лучшем и худшем случаях;
- провести тестирование по методу черного ящика для реализации указанного алгоритма;
- 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 рассмотрен алгоритм Кнута — Морриса — Пратта.

1.1 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.

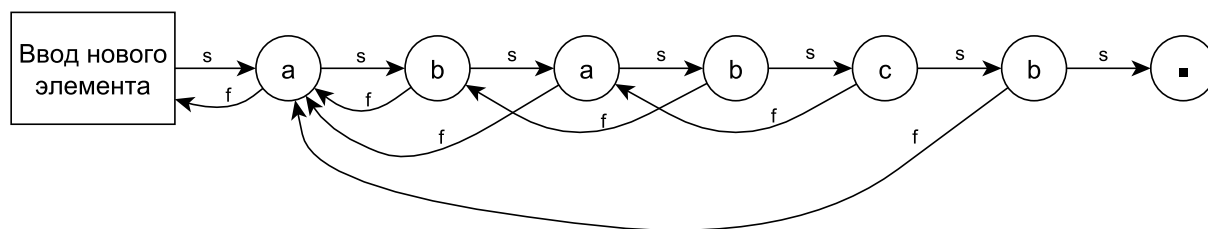


Рисунок 1.1 – Автомат для определения величин смещения

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

1.2 Вывод

В данном разделе был теоретически разобран алгоритм Кнута — Морриса — Пратта.

К разрабатываемой программе предъявляются следующие требования:

1. Программа должна предоставлять функциональность поиска бинарной подстроки в строке;

2. Реализуемое ПО будет работать в одном режиме — пользовательском, в котором можно вывести результат работы алгоритма;
3. В качестве входных данных в программу будет подаваться файл, содержащий бинарную строку, в которой проводится поиск, также реализовано меню для вызова алгоритма и замеров времени. Программа должна корректно обрабатывать случай ввода подстроки, которой нет в исходной строке;

2 Конструкторская часть

В этом разделе будет представлена схема алгоритма Кнута — Морриса — Пратта, а также рассчитана трудоемкость алгоритма в лучшем и худшем случаях.

2.1 Разработка алгоритмов

На рисунках 2.1 – 2.2 представлена схема алгоритма Кнута — Морриса — Пратта.

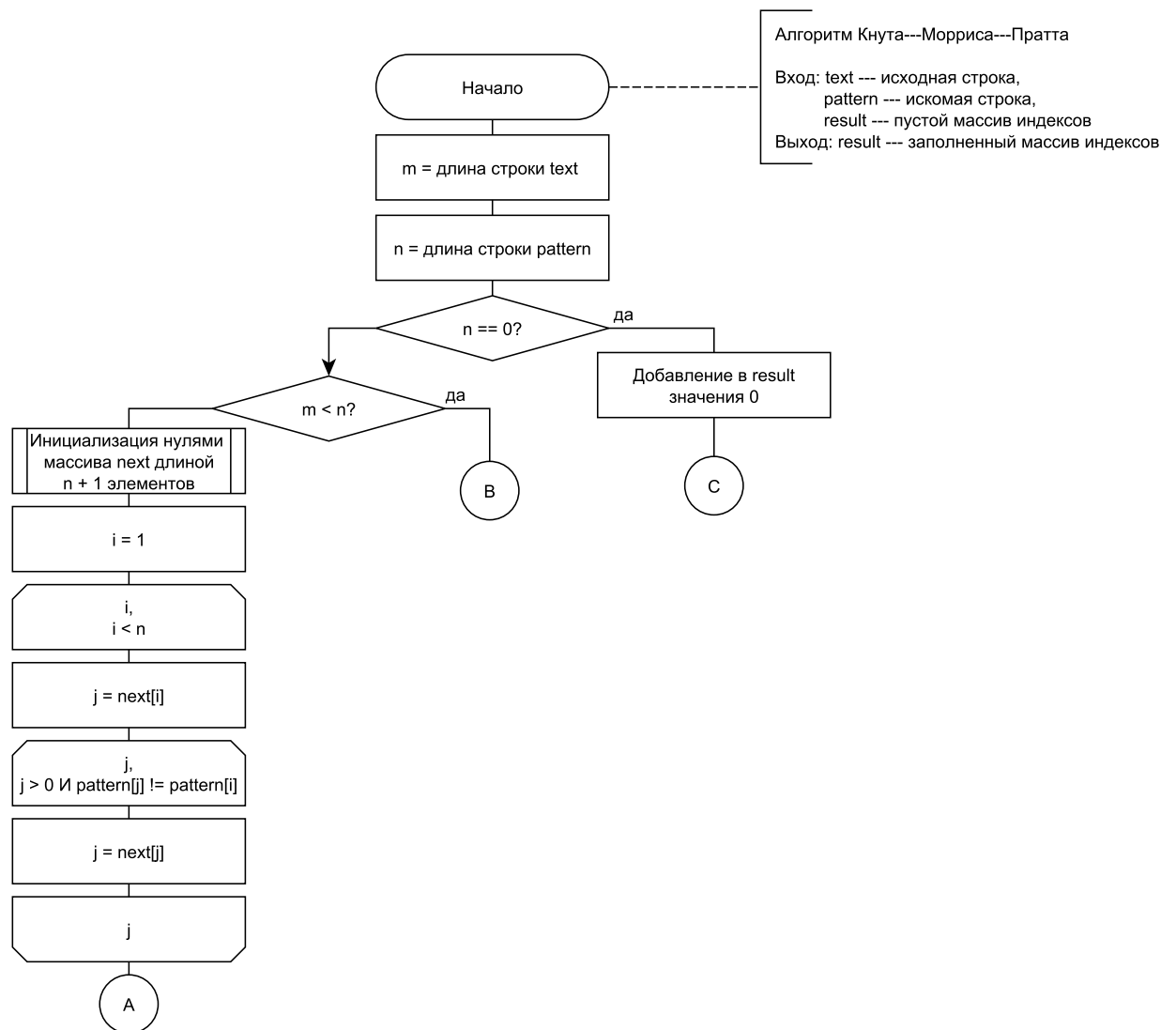


Рисунок 2.1 – Схема алгоритма Кнута — Морриса — Пратта (часть 1)

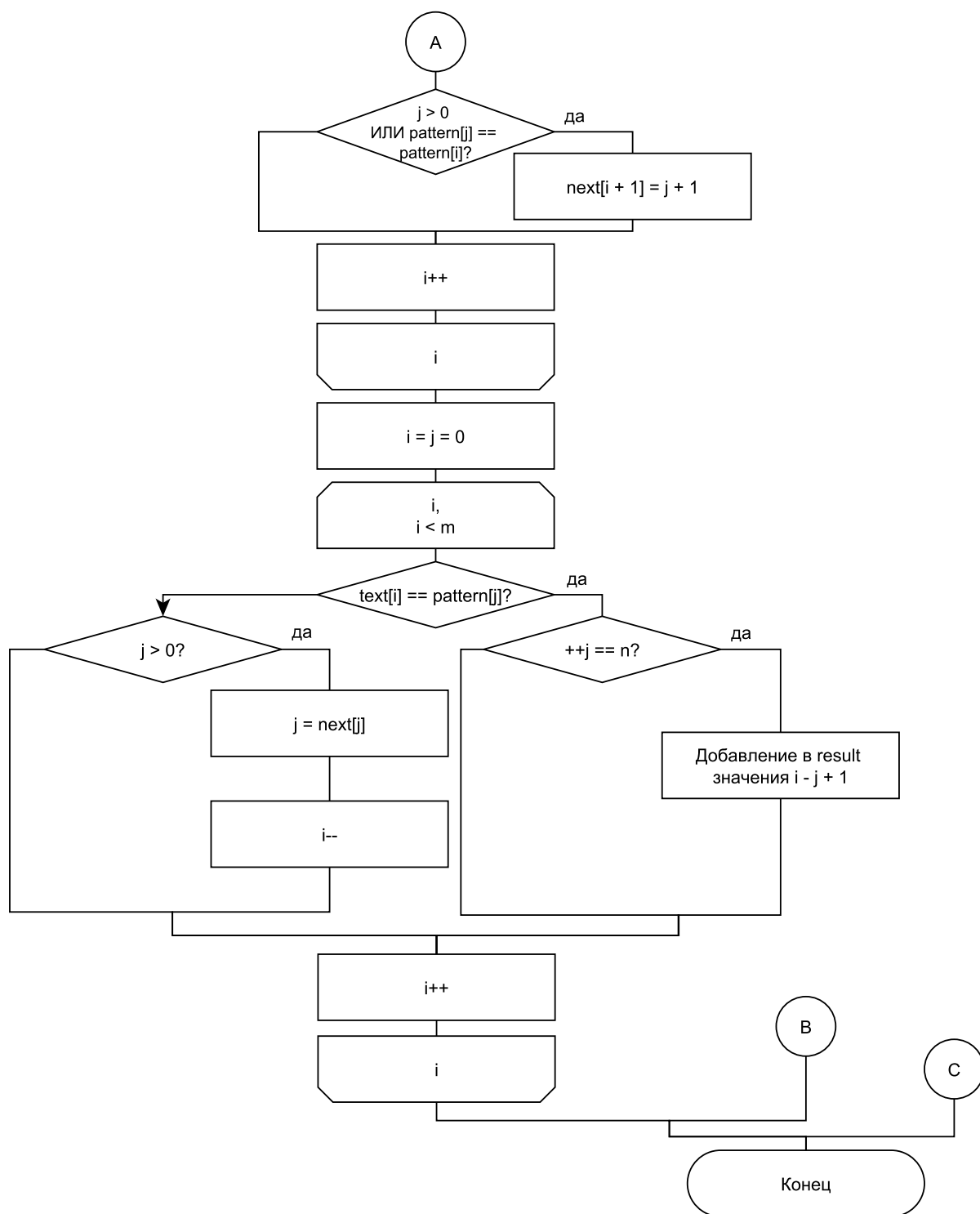


Рисунок 2.2 – Схема алгоритма Кнута — Морриса — Пратта (часть 2)

2.2 Классы эквивалентности тестирования

Для тестирования выделены следующие классы эквивалентности:

1. ввод подстроки, которой нет в исходной строке;
2. ввод пустой подстроки;
3. ввод подстроки, содержащей повторяющуюся часть;
4. ввод подстроки, состоящей из одного символа;
5. ввод произвольной подстроки.

2.3 Модель вычислений

1. Трудоемкость базовых операций.

Следующие операторы имеют трудоемкость 1:

$+, -, + =, - =, =, ==, !=, > =, < =, >, < ,$

$>>, <<, [], \&, |, \&\&, ||, ++, --$

Следующие операторы имеют трудоемкость 2:

$*, /, \%, * =, / =$

2. Условный оператор.

Для конструкций вида:

```
1      if (условие)
2      {
3          Блок 1;
4      }
5      else
6      {
7          Блок 2;
8      }
```


Пусть трудоемкость блока 1 — f_1 , блока 2 — f_2 . Пусть также трудоемкость условного перехода — 0.

Тогда трудоемкость условного оператора:

$$f_{if} = f_{\text{вычисления условия}} + \begin{cases} \min(f_1, f_2), \text{ лучший случай} \\ \max(f_1, f_2), \text{ худший случай} \end{cases} \quad (2.1)$$

3. Трудоемкость циклов.

Трудоемкость циклов вычисляется по следующей формуле:

$$f_{\text{цикла}} = f_{\text{инициализации}} + f_{\text{сравнения}} + \\ + M_{\text{шагов}} \cdot (f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \quad (2.2)$$

2.4 Трудоемкость алгоритма

Рассчитаем трудоемкость алгоритма Кнута — Морриса — Пратта в лучшем и худшем случаях.

В таблице 2.1 представлена построчная оценка трудоемкости.

Таблица 2.1 – Построчная оценка трудоемкости

Строка кода	Вес
<code>int m = text.length();</code>	2
<code>int n = pattern.length();</code>	2
<code>if (n == 0)</code>	1
<code>result.insert(0);</code>	1
<code>return;</code>	0
<code>if (m < n)</code>	1
<code>return;</code>	0
<code>int next[n + 1];</code>	1
<code>for (int i = 0; i < n + 1; i++)</code>	4
<code>next[i] = 0;</code>	2
<code>for (int i = 1; i < n; i++)</code>	3
<code>int j = next[i];</code>	2
<code>while (j > 0 && pattern[j] != pattern[i])</code>	5
<code>j = next[j];</code>	2
<code>if (j > 0 pattern[j] == pattern[i])</code>	5
<code>next[i + 1] = j + 1;</code>	4
<code>for (int i = 0, j = 0; i < m; i++)</code>	4
<code>if (text[i] == pattern[j])</code>	3
<code>if (++j == n)</code>	2
<code>result.insert(start_pos + i - j + 1);</code>	4
<code>else if (j > 0)</code>	1
<code>j = next[j];</code>	2
<code>i- -;</code>	1

Лучший случай: длина искомой подстроки равна 0. Трудоемкость:

$$f = O(6) = O(1) \quad (2.3)$$

Худший случай: в исходной строке нет искомой подстроки. Трудоемкость:

$$f = O(2m + 2n - 3) = O(m + n) \quad (2.4)$$

2.5 Вывод

В данном разделе была представлена схема алгоритма Кнута — Морриса — Пратта и рассчитана трудоемкость в лучшем и худшем случаях.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги реализации алгоритма Кнута — Морриса — Пратта.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования `C++` [2].

3.2 Описание используемых типов данных

При реализации будут использованы следующие типы и структуры данных:

- `std::set` для результирующего набора индексов;
- `char *` для исходной строки;
- `std::string` для подстроки.

3.3 Сведения о модулях программы

Программа состоит из одного модуля:

- `main.cpp` — файл, содержащий реализацию алгоритма Кнута — Морриса — Пратта.

3.4 Реализация алгоритмов

В листинге 3.1 представлена реализация алгоритма Кнута — Морриса — Пратта для бинарных строк.

Листинг 3.1 – Реализация алгоритма Кнута — Морриса — Пратта для
бинарных строк

```
1 void KMP(int start_pos, string text, string pattern, std::set<int>
   &result)
2 {
3     int m = text.length();
4     int n = pattern.length();
5     if (n == 0)
6     {
7         result.insert(0);
8         return;
9     }
10    if (m < n)
11        return;
12
13    // next[i] сохраняет индекс следующего лучшего частичного совпад
        ения
14    int next[n + 1];
15    for (int i = 0; i < n + 1; i++) {
16        next[i] = 0;
17    }
18
19    for (int i = 1; i < n; i++)
20    {
21        int j = next[i];
22        while (j > 0 && pattern[j] != pattern[i]) {
23            j = next[j];
24        }
25        if (j > 0 || pattern[j] == pattern[i]) {
26            next[i + 1] = j + 1;
27        }
28    }
29
30    for (int i = 0, j = 0; i < m; i++)
31    {
32        if (text[i] == pattern[j])
33        {
34            if (++j == n) {
35                result.insert(start_pos + i - j + 1);
36            }
37        }
```

```

38     else if (j > 0)
39     {
40         j = next[j];
41         i--;    // так как 'i' будет увеличен на следующей итера
                  ции
42     }
43 }
44 }

```

3.5 Функциональные тесты

В таблице 3.1 приведены тесты для функции, реализующей алгоритм Кнута — Морриса — Пратта. Тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Исходная строка	Искомая подстрока	Реализация алгоритма Кнута — Морриса — Пратта
abcd	"пустая строка"	0
abab	cd	«подстрока не найдена»
abababcb	ababcb	2
abcabd	abc	0
ababcacab	ab	0, 2, 7

3.6 Вывод

Была представлена реализация алгоритма Кнута — Морриса — Пратта, который был описан в предыдущем разделе. Также в данном разделе была приведена информация о выбранных средствах для разработки алгоритма.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы.

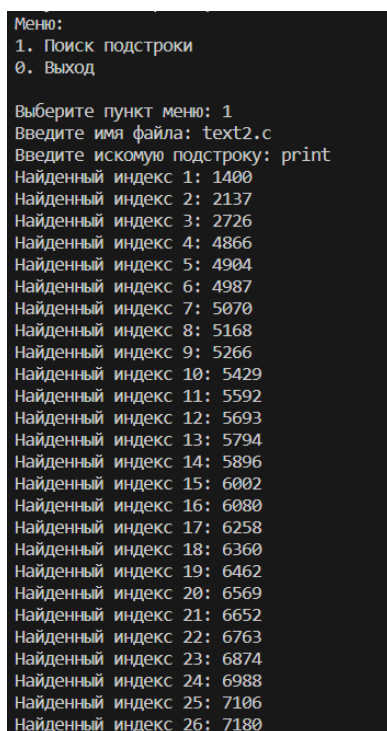
4.1 Технические характеристики

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 [1];
- 6 физических ядер, 12 логических ядер [1].

4.2 Демонстрация работы программы

На рисунке 4.1 представлен результат работы программы.



```
Меню:  
1. Поиск подстроки  
0. Выход  
  
Выберите пункт меню: 1  
Введите имя файла: text2.c  
Введите искомую подстроку: print  
Найденный индекс 1: 1400  
Найденный индекс 2: 2137  
Найденный индекс 3: 2726  
Найденный индекс 4: 4866  
Найденный индекс 5: 4904  
Найденный индекс 6: 4987  
Найденный индекс 7: 5070  
Найденный индекс 8: 5168  
Найденный индекс 9: 5266  
Найденный индекс 10: 5429  
Найденный индекс 11: 5592  
Найденный индекс 12: 5693  
Найденный индекс 13: 5794  
Найденный индекс 14: 5896  
Найденный индекс 15: 6002  
Найденный индекс 16: 6080  
Найденный индекс 17: 6258  
Найденный индекс 18: 6360  
Найденный индекс 19: 6462  
Найденный индекс 20: 6569  
Найденный индекс 21: 6652  
Найденный индекс 22: 6763  
Найденный индекс 23: 6874  
Найденный индекс 24: 6988  
Найденный индекс 25: 7106  
Найденный индекс 26: 7180
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Вывод

В данном разделе был приведен пример работы программы.

Заключение

В результате было получено, что в лучшем случае реализация алгоритма Кнута — Морриса — Пратта работает за $O(1)$, а в худшем за $O(n + m)$, где n — длина искомой подстроки, m — длина исходной строки.

Цель, которая была поставлена в начале лабораторной работы была достигнута: алгоритм Кнута — Морриса — Пратта был применен для поиска подстрок в бинарных строках. Для достижения поставленной цели были выполнены все задачи:

- описан алгоритм Кнута — Морриса — Пратта;
- реализован указанный алгоритм;
- рассчитана трудоемкость в лучшем и худшем случаях;
- проведено тестирование по методу черного ящика для реализации указанного алгоритма;
- описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

Список используемых источников

1. AMD Ryzen 5 5500U. — URL: <https://www.amd.com/en/products/apu/amd-ryzen-5-5500u> (дата обр. 01.12.2023).
2. C++ documnetation. — URL: <https://cplusplus.com/> (дата обр. 01.12.2023).
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. 01.12.2023).