

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №4**  
**по дисциплине «ПОСТРОЕНИЕ И АНАЛИЗ АЛГОРИТМОВ»**  
**ТЕМА: Алгоритм Кнута-Морриса-Пратта**

Студентка гр. 8383

Максимова А.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

## **Цель работы.**

Изучить алгоритм Кнута-Морриса-Пратта, используемый для поиска подстроки в тексте, и реализовать его в собственной программе для решения поставленных задач.

## **Постановка задач.**

1. Реализуйте алгоритм КМП и с его помощью для заданных шаблона  $P(|P| \leq 15000)$  и текста  $T (|T| \leq 5000000)$  найдите все вхождения  $P$  в  $T$ .

### **Вход:**

Первая строка –  $P$

Вторая строка –  $T$

### **Выход:**

индексы начал вхождений  $P$  в  $T$ , разделенных запятой, если  $P$  не входит в  $T$ , то вывести  $-1$ .

### **Sample Input:**

ab

abab

### **Sample Output:**

0,2

2. Заданы две строки  $A (|A| \leq 5000000)$  и  $B (|B| \leq 5000000)$ .

Определить, является ли  $A$  циклическим сдвигом  $B$  (это значит, что  $A$  и  $B$  имеют одинаковую длину и  $A$  состоит из суффикса  $B$ , склеенного с префиксом  $B$ ). Например, defabc является циклическим сдвигом abcdef.

### **Вход:**

Первая строка -  $AA$

Вторая строка -  $BB$

### **Выход:**

Если A является циклическим сдвигом B, индекс начала строки B в A, иначе вывести  $-1$ . Если возможно несколько сдвигов вывести первый индекс.

### **Sample Input:**

defabc

abcdef

### **Sample Output:**

3

**Вар. 1.** Подготовка к распараллеливанию: работа по поиску разделяется на  $k$  равных частей, пригодных для обработки  $k$  потоками (при этом длина образца гораздо меньше длины строки поиска).

### **Основные теоретические положения.**

*Задача поиска в строке.* Пусть есть некоторый текст  $T$  и образ (шаблон)  $R$ . Необходимо найти первое (или все) вхождение этого шаблона в указанном тексте.

#### Способы решения.

*Прямой поиск:* сравниваем первый символ образа с первым символом текста, в случае совпадения, сравниваем следующие символы, иначе сдвигаем шаблон на один символ вправо. Процесс продолжается до тех пор, пока не будет найдено вхождение (или все вхождения, в зависимости от постановки задачи) или пока не закончится указанный текст, что будет означать, что вхождений образа нет.

*Алгоритм Кнута-Морриса-Пратта (КМП):* также сравнивает символ из текста с символом из образа и в случае несовпадения позволяет делать сдвиги больше, чем на один.

*Префикс-функция от строки и позиции в ней* – длина наибольшего собственного префикса подстроки, который одновременно является суффиксом этой подстроки.

*Префикс* – это символы, которые стоят в начале образа (шаблона, подстроки).

Шаблон (образ): a b c d e f g

Префикс длины 1: **a** b c d e f g

Префикс длины 2: **a b** c d e f g

Префикс длины 3: **a b c** d e f g

Префикс длины 4: **a b c d** e f g

Префикс длины 5: **a b c d e** f g

Префикс длины 6: **a b c d e f** g

Префикс длины 7 не рассматривается  
– иначе алгоритм теряет смысл

Рисунок 1 - Префикс

*Суффикс* – это символы, которые стоят в конце образа. Рисунок в данном случае отличается только тем, что берем символы с конца.

### Описание структур данных.

class KMP, используемый для реализации алгоритма Кнута-Морриса-Пратта.

\* см. приложение А

### Поля:

- int sizeTemp; – целочисленная переменная, используемая для хранения размера (длины) шаблона
- int sizeText; – целочисленная переменная, используемая для хранения размера (длины) текста

- `int task;` – целочисленная переменная, используемая для хранения номера команды: 0 – поиск всех вхождений шаблона в текст, 1 – определение циклического сдвига с указанием вхождения
- `int quantityBlock;` – целочисленная переменная, используемая для хранения количества блоков (потоков), на которые необходимо разделить введенный пользователем текст
- `int sizeBlock;` – целочисленная переменная, используемая для хранения минимального размера блока, то есть отношение длины текста к количеству блоков
- `int sizeBlockSk;` – целочисленная переменная, используемая для хранения максимального размера блока, то есть сумма минимального размера блока и размера “пересечений”, которые необходимы для того, чтобы не пропустить вхождение шаблона в текст при его разделении на части
- `std::string templP;` – переменная, для хранения шаблона
- `std::string textT;` – переменная, для хранения текста
- `std::vector <int> prefixFunc;` – вектор, для хранения префикс-функции, используемой в КМП
- `std::vector <int> indexEntry;` – вектор, используемый для хранения целочисленных значений – индексов вхождений шаблона в текст
- `std::vector <std::string> block;` – вектор, используемый для хранения блоков

## **Методы:**

- `void setTemp1P(std::string P);` – метод, принимающий введенный пользователем шаблон и устанавливающий его значение в поле `temp1P`. Метод ничего не возвращает.
  - `void setTextT(std::string T);` – метод, принимающий введенный пользователем текст и устанавливающий его значение в поле `textT`. Метод ничего не возвращает.
  - `void setQuantBlock(int quantBl);` – метод, принимающий целочисленное значение количества блоков, на которые необходимо разделить текст, и устанавливающий данное значение в поле `quantityBlock`.
  - `int checkFirst(bool flag);` – метод, принимающий булевскую переменную – флаг. Используется для проверки данных, необходимых для действия: поиск вхождений шаблона в текст. Флаг используется для вывода ошибок, если они присутствуют, или игнорирования этого вывода. Метод возвращает целое число, меньшее единицы, в случае возникновения ошибки, единицу в случае, если данные введены корректно.
  - `int checkSecond(std::string T, std::string P, bool flag);` – метод, принимающий булевскую переменную – флаг, исходный текст и шаблон. Используется для проверки данных, необходимых для действия: определение циклического сдвига. Флаг используется для вывода ошибок, если они присутствуют, или игнорирования этого вывода. Метод возвращает целое число, меньшее единицы, в случае возникновения ошибки, единицу в случае, если данные введены корректно.
- \* Данный метод принимает шаблон и текст, так как эти значения еще не установлены в поля класса, по причине того, что после установки значение текста увеличится вдвое – для удобства поиска циклического сдвига.

- `bool find(int elem);` – метод, принимающий целочисленную переменную – индекс вхождения, который необходимо добавить в вектор результатов. Метод используется для проверки: не был ли индекс уже включен в результат. Возвращает булевское значение `false`, если индекс встречается впервые, иначе `true`.
- \*Так как методы: `void trainingText();` `void createPrefixFunc();` `void patternSearch(std::string block, int index);` и `void alghorithmKMP();` используются для реализации алгоритма, то их назначение представлено в Описании Алгоритма.
- `void readData(int task_);` – метод, принимающий целочисленное значение номера команды: 0 или 1. Используется для чтения данных, введенных пользователем, проверки их корректности, установки их значений в поля класса. Так же из метода вызывается алгоритм КМП. Метод ничего не возвращает.
  - `void printText();` – метод ничего не принимает, используется для печати текста, в котором осуществляется поиск вхождений шаблона. Необходим для вывода промежуточных данных. Метод ничего не возвращает.
  - `void printBlock(int index);` – метод, принимающий целочисленное значение номера блока, который необходимо вывести. Используется для печати в консоль промежуточных данных. Метод ничего не возвращает.
  - `void printPattern();` – метод ничего не принимает, используется для печати значения шаблона. Используется для вывода промежуточных данных. Метод ничего не возвращает.
  - `void printPrefixFunc();` – метод, ничего не принимающий на вход. Необходим для печати префикс-функции для вывода промежуточных данных. Метод ничего не возвращает.
  - `void printAnswer();` – метод ничего не принимает, используется

для печати результатов, полученных с помощью алгоритма КМП.  
Метод ничего не возвращает.

## Описание алгоритма.

Для поэтапной реализации алгоритма Кнута-Морриса-Пратта с целью поиска подстроки в тексте были написаны итеративные методы

```
void trainingText(); void createPrefixFunc(); void
patternSearch(std::string block, int index); и void
algorithmKMP().
```

## Этап 1. Подготовка к распараллеливанию.

### Алгоритм

#### Подготовка к распараллеливанию

\*работа по поиску разделяется на k равных частей, пригодных для обработки k потоками

k - количество блоков

sizePattern - размер шаблона

sizeBlock - размер блока

sizeBlock = длина текста / k

Тогда, размер сканируемого блока:

sizeBlockSk = sizeBlock + sizePattern - 1

Ограничения:

k > 0

sizeBlockSk >= sizePattern

«Хвост строки» не  
должен быть  
меньше sizePattern

sizeText / k >= sizePattern - 1

=>

sizeText/(sizePattern-1) >= k

**k <= 5**

нарушение  
условия

Исходные данные:

Текст: A B C D G R E E Y G R E E N O L O L R A A

sizeText = 21

Шаблон: G R E E N

sizePattern = 5

Пусть k = 3:

sizeBlock = 21 / 3 = 7

sizeBlockSk = 7 + (5 - 1) = 11

Полученные блоки:

1) A B C D G R E E Y G R

2) E Y G R E E N O L O L

3) O L O L R A A

Пересечения -  
для того, чтобы  
не пропустить  
изначальное  
вхождение  
шаблона в текст

искомое вхождение

Пусть k = 5:

sizeBlock = 21 / 5 = 4

sizeBlockSk = 4 + (5 - 1) = 8

Полученные блоки:

1) A B C D G R E E

2) G R E E Y G R E

3) Y G R E E N O L

4) E N O L O L R A

5) O L R A A

Пусть k = 8:

sizeBlock = 21 / 8 = 2

sizeBlockSk = 2 + (5 - 1) = 6

Полученные блоки:

1) A B C D G R

2) C D G R E E

3) G R E E Y G

4) E E Y G R E

5) Y G R E E N

6) R E E N O L

7) E N O L O L

8) O L O L R A

9) O L R A A

получили 9 блоков  
вместо 8

Paint X Lite

Рисунок 2 – Пример распараллеливания

## Реализация

Для реализации распараллеливания был написан метод void trainingText(); Метод ничего не принимает, необходим для разделения исходного текста на блоки, количество которых определяется пользователем. Метод ничего не возвращает.

## **Этап 2. Построение префикс-функции.**

Алгоритм можно представить в виде следующей последовательности шагов:

**Шаг 1 - Подготовка:** Создание одномерного массива (вектора), размер которого равен длине шаблона. После заполнения массива, префикс-функция для  $i$ -ого символа образа будет возвращать некоторое значение, используемое в алгоритме КМП.

Для первого символа образа в массив префикс-функции записывается значение ноль. Рассматриваемые символы обозначим за  $j$  и  $i$  (изначально это соответственно индексы 0 и 1).

\*Рассматриваем только те суффиксы и префиксы, длины которых равны.

## **Шаг 2 - Сравнение символов.**

Если символы, на которые указывают  $i$  и  $j$  не равны и  $j$  указывает на начало шаблона, то записываем в префикс-функцию от  $i$ -ого значения – ноль и сдвигаем  $i$  на единицу.

Иначе присваиваем индексу  $j$  значение префикс-функции предыдущего символа, на который указывает  $j$ .

Если символы равны, тогда в префикс-функцию от  $i$ -го элемента записываем значение  $j + 1$ , и сдвигаемся по обоим индексам.

## **Шаг 3 – Окончание.**

Повторяем шаг 2 до тех пор, пока не дойдем до конца шаблона.

Пример работы алгоритма см. на рис. 3.

## Реализация

Для реализации описанного выше алгоритма был написан метод void `createPrefixFunc()`. Метод ничего не принимает и ничего не возвращает, вычисляет и устанавливает значение префикс-функции, в соответствии с алгоритмом.

*Иллюстрация работы алгоритма:*

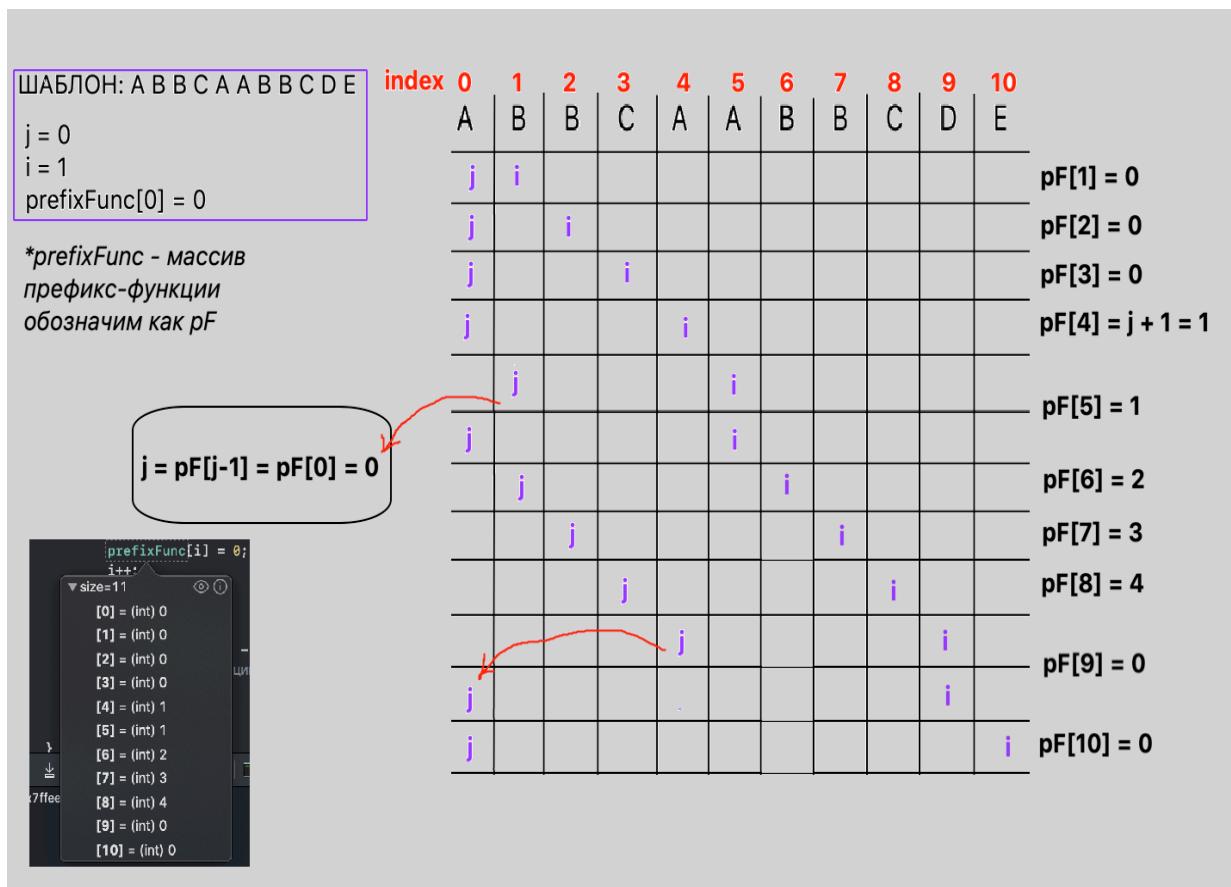


Рисунок 3 – Формирование префикс-функции

### Этап 3. Поиск образа в строке.

Алгоритм можно представить в виде следующей последовательности шагов:

**Шаг 1.** Заведем два “указателя”: первый указывает на индекс рассматриваемого символа текста, второй – шаблона. Оба указывают на начало.

**Шаг 2.** Сравниваем текущие символы.

**Шаг 2.1.** Если совпадают, смещаем указатели на единицу и:

**Шаг 2.1.1.** Если “указатель” шаблона оказался в конце, то значит, нашли вхождение – фиксируем его в массиве результатов и переходим на шаг 3.

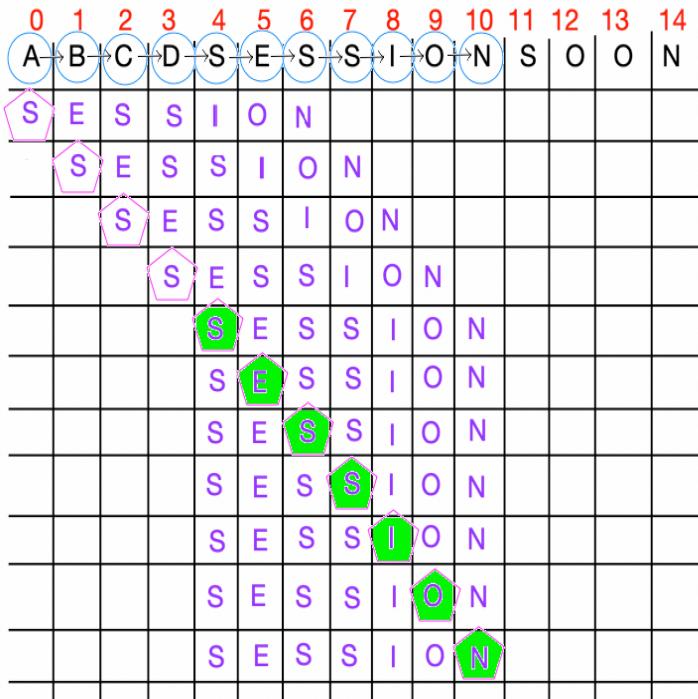
**Шаг 2.2.** Если символы не равны и:

**Шаг 2.2.1.** Указатель на шаблон находится в начале, то присваиваем ему значение префикс-функции от предыдущего для данного значения.

**Шаг 2.2.2.** Иначе перемещаем указатель на текст на единицу.

**Шаг 3.** Возвращаемся на шаг 2, если еще не достигли конца текста.

*Иллюстрация работы алгоритма:*

Поиск шаблона в тексте  
Текст: ABCDSESSIONSOON  
Шаблон: SESSION  
  
○ - рассматриваемый символ текста  
○ - рассматриваемый символ шаблона

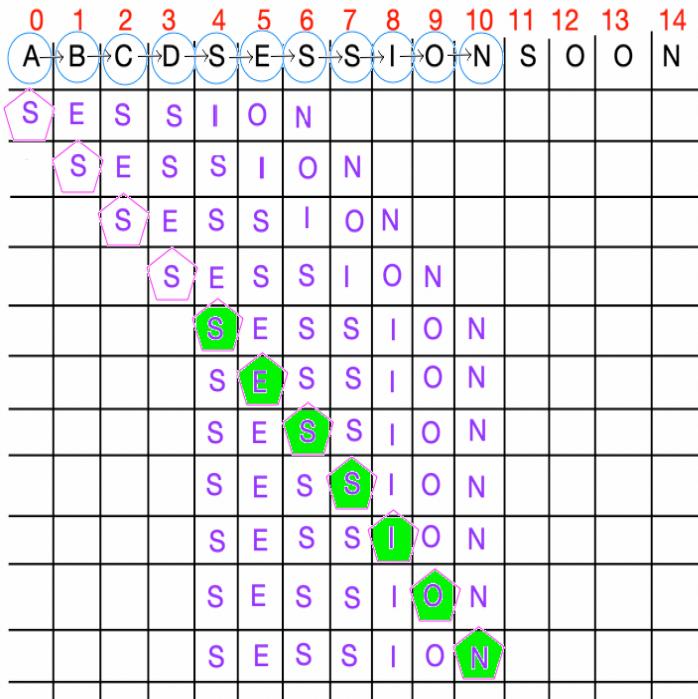


Рисунок 4 – Пример начала работы алгоритма поиска шаблона

## Реализация

Для реализации описанного выше алгоритма был написан метод `void patternSearch(std::string block, int index);`. Метод принимает обрабатываемый блок типа `string` и его порядковый номер, значение которого хранится в целочисленной переменной. Метод ничего не возвращает.

## **Этап 4. Сборка.**

Для объединения описанных выше этапов используется метод `void alghorithmKMP()`. Кроме того, метод используется для последовательной передачи блоков, которые необходимо обработать в основной метод, осуществляющий поиск образа в тексте. После обращения ко всем необходимым методам, происходит вызов функции печати результата. Метод ничего не принимает и ничего не возвращает.

## **Оценка сложности алгоритма.**

Сложность алгоритма по времени для поиска образа длины  $m$  в тексте длины  $n$  можно оценить как  $O(m + n)$ , так как алгоритм можно разделить на построение префикс-функции и поиск вхождений: сложность вычисления префикс-функции по образу равна  $O(m)$ , сложность прохождения по всему тексту равна  $O(n)$ .

Сложность алгоритма по памяти также равна  $O(m + n)$ , так как в процессе работы алгоритма мы храним исходный шаблон  $O(m)$ , префикс функцию  $O(m)$  и текст  $O(n)$ , константа отбрасывается.

## Тестирование: Поиска образа в строке.

Подробный тест.

Входные данные.

```
Введите а, если вы хотите вывести все вхождения шаблона Р в текст Т.  
Введите b, если вы хотите определить является ли А циклическим сдвигом в В.  
а  
Пожалуйста, введите шаблон.  
abcd  
Пожалуйста, введите текст.  
ffgabcdkabc  
Введите количество блоков, на которое хотите разделить текст.  
2
```

Выходные данные.

**Этап 1 – Разделение текста на k = 2 блоков**

**Исходный текст: ffgabcdkabc**

**Количество блоков k = 2**

**Максимальный размер блока: 8**

**Блок 1: ffgabcdk**

**Блок 2: cdkabc**

**Этап 2 – Формирование массива префикс–функции**

**Обрабатываемый шаблон: abcd**

.....

**Обработка: шаблон[0] = а и шаблон[1] = б**

**Символы не совпадают!**

**prefixFunc[1] = 0**

.....

**Обработка: шаблон[0] = а и шаблон[2] = с**

**Символы не совпадают!**

**prefixFunc[2] = 0**

.....

**Обработка: шаблон[0] = а и шаблон[3] = д**

**Символы не совпадают!**

**prefixFunc[3] = 0**

**Полученная префикс–функция:**

**Шаблон[0] = а --> prefixFunc[0] = 0**

**Шаблон[1] = б --> prefixFunc[1] = 0**

**Шаблон[2] = с --> prefixFunc[2] = 0**

**Шаблон[3] = д --> prefixFunc[3] = 0**

**Этап 3 – Поиск вхождений шаблона в текст**

**Обрабатываемый блок: 1) ffgabcdk**

**Шаблон :abcd**

**Блок: ffgabcdk**

**Символы для сравнения: шаблон[0] = а и блок[0] = f**

**Символы не совпадают!**

**Символы для сравнения: шаблон[0] = а и блок[1] = f**

**Символы не совпадают!**

**Символы для сравнения: шаблон[0] = а и блок[2] = g**

**Символы не совпадают!**

**Символы для сравнения: шаблон[0] = а и блок[3] = a**

**Символы совпадают!**

**Символы для сравнения: шаблон[1] = b и блок[4] = b**

**Символы совпадают!**

**Символы для сравнения: шаблон[2] = c и блок[5] = c**

**Символы совпадают!**

**Символы для сравнения: шаблон[3] = d и блок[6] = d**

**Символы совпадают!**

**!!!Нашли вхождение шаблона в текст: 3**

**Символы для сравнения: шаблон[4] = и блок[7] = k**

**Символы не совпадают!**

**Символы для сравнения: шаблон[0] = a и блок[7] = k**

**Символы не совпадают!**

**Обрабатываемый блок: 2) cdkabc**

**Шаблон :abcd**

**Блок: cdkabc**

**Символы для сравнения: шаблон[0] = a и блок[0] = c**

**Символы не совпадают!**

**Символы для сравнения: шаблон[0] = a и блок[1] = d**

**Символы не совпадают!**

All Output ◇

**Символы для сравнения: шаблон[0] = а и блок[2] = к  
Символы не совпадают!**

**Символы для сравнения: шаблон[0] = а и блок[3] = а  
Символы совпадают!**

**Символы для сравнения: шаблон[1] = б и блок[4] = б  
Символы совпадают!**

**Символы для сравнения: шаблон[2] = с и блок[5] = с  
Символы совпадают!**

**РЕШЕНИЕ:**

**Вхождения шаблона в текст найдены: 3  
Program ended with exit code: 0**

*Таблица тестирования.*

Входные данные	Выходные данные
<p><b>Пожалуйста, введите шаблон. abc</b></p> <p><b>Пожалуйста, введите текст. abcabcabca</b></p> <p><b>Введите количество блоков, на 4</b></p>	<p><b>РЕШЕНИЕ: Вхождения шаблона в текст найдены: 0,3,6,9,12,15 Program ended with exit code: 0</b></p>
<p><b>Пожалуйста, введите шаблон. ab</b></p> <p><b>Пожалуйста, введите текст. abab</b></p> <p><b>Введите количество блоков, на 3</b></p>	<p><b>РЕШЕНИЕ: Вхождения шаблона в текст найдены: 0,2 Program ended with exit code: 0</b></p>



<p><b>Пожалуйста, введите шаблон.</b> jkdsh</p> <p><b>Пожалуйста, введите текст.</b> sjadhhdsjhsj</p> <p><b>Введите количество блоков, на</b> 0</p>	<p><b>Некорректное количество блоков.</b> <b>Попробуйте снова.</b></p>
<p><b>Пожалуйста, введите шаблон.</b> a</p> <p><b>Пожалуйста, введите текст.</b> lalalalalalalalala</p> <p><b>Введите количество блоков, на</b> 3</p>	<p><b>РЕШЕНИЕ:</b> Вхождения шаблона в текст найдены: 1,3,5,7,9,11,13,15,17 Вхождения не найдены: 0</p>
<p><b>Пожалуйста, введите шаблон.</b> abcd</p> <p><b>Пожалуйста, введите текст.</b> abcrabhrabedaabbcd</p> <p><b>Введите количество блоков,</b> 4</p>	<p><b>РЕШЕНИЕ:</b> Вхождения шаблона в текст не найдены: -1</p>

### Тестирование: Определение циклического сдвига.

*Подробный тест.*

Входные данные.

**Введите а, если вы хотите вывести все вхождения шаблона Р в текст Т.**  
b

**Введите b, если вы хотите определить является ли А циклическим сдвигом в В.**  
b

**Пожалуйста, введите А.**  
defabc

**Пожалуйста, введите В.**  
abcdef

**Введите количество блоков, на которое хотите разделить текст.**  
3

**Некорректное количество блоков.**  
2

## Выходные данные.

**Этап 1 – Разделение текста на k = 2 блоков**  
Исходный текст: defabcdefab  
Количество блоков k = 2  
Максимальный размер блока: 11  
Блок 1: defabcdefab  
Блок 2: defabc

**Этап 2 – Формирование массива префикс-функции**  
Обрабатываемый шаблон: abcdef  
.....  
Обработка: шаблон[0] = a и шаблон[1] = b  
Символы не совпадают!  
prefixFunc[1] = 0  
.....  
Обработка: шаблон[0] = a и шаблон[2] = c  
Символы не совпадают!  
prefixFunc[2] = 0  
.....  
Обработка: шаблон[0] = a и шаблон[3] = d  
Символы не совпадают!  
prefixFunc[3] = 0  
.....  
Обработка: шаблон[0] = a и шаблон[4] = e  
Символы не совпадают!  
prefixFunc[4] = 0  
.....  
Обработка: шаблон[0] = a и шаблон[5] = f  
Символы не совпадают!  
prefixFunc[5] = 0

**Полученная префикс-функция:**

Шаблон[0] = a --> prefixFunc[0] = 0  
Шаблон[1] = b --> prefixFunc[1] = 0  
Шаблон[2] = c --> prefixFunc[2] = 0  
Шаблон[3] = d --> prefixFunc[3] = 0  
Шаблон[4] = e --> prefixFunc[4] = 0  
Шаблон[5] = f --> prefixFunc[5] = 0

**Этап 3 – Поиск вхождений шаблона в текст**

**Обрабатываемый блок: 1) defabcdefab**

**Шаблон :abcdef**

**Блок: defabcdefab**

**Символы для сравнения: шаблон[0] = а и блок[0] = д**

**Символы не совпадают!**

**Символы для сравнения: шаблон[0] = а и блок[1] = е**

**Символы не совпадают!**

**Символы для сравнения: шаблон[0] = а и блок[2] = ф**

**Символы не совпадают!**

**Символы для сравнения: шаблон[0] = а и блок[3] = а**

**Символы совпадают!**

**Символы для сравнения: шаблон[1] = б и блок[4] = б**

**Символы совпадают!**

**Символы для сравнения: шаблон[2] = с и блок[5] = с**

**Символы совпадают!**

**Символы для сравнения: шаблон[3] = д и блок[6] = д**

**Символы совпадают!**

**Символы для сравнения: шаблон[4] = е и блок[7] = е**

**Символы совпадают!**

**Символы для сравнения: шаблон[5] = ф и блок[8] = ф**

**Символы совпадают!**

**!!!Нашли вхождение шаблона в текст: 3**

**Символы для сравнения: шаблон[6] = и блок[9] = а**

**Символы не совпадают!**

**Символы для сравнения: шаблон[0] = а и блок[9] = а**

**Символы совпадают!**

Символы для сравнения: шаблон[1] = b и блок[10] = b  
Символы совпадают!

Обрабатываемый блок: 2) defabc  
Шаблон :abcdef  
Блок: defabc

Символы для сравнения: шаблон[0] = a и блок[0] = d  
Символы не совпадают!

Символы для сравнения: шаблон[0] = a и блок[1] = e  
Символы не совпадают!

Символы для сравнения: шаблон[0] = a и блок[2] = f  
Символы не совпадают!

Символы для сравнения: шаблон[0] = a и блок[3] = a  
Символы совпадают!

Символы для сравнения: шаблон[1] = b и блок[4] = b  
Символы совпадают!

Символы для сравнения: шаблон[2] = c и блок[5] = c  
Символы совпадают!

#### РЕШЕНИЕ:

А является циклическим сдвигом В. Индекс начала строки В в А: 3

Таблица тестирования.

Входные данные	Выходные данные
Пожалуйста, введите А. bbb Пожалуйста, введите В. bbb	bbb Введены два эквивалентных текста 0

<p><b>Пожалуйста, введите А.</b> lalala <b>Пожалуйста, введите В.</b> hahahahah</p>	<p><b>Ошибка:</b> Длина А должна быть равна длине В -1</p>
<p><b>Пожалуйста, введите А.</b> hmaaaa <b>Пожалуйста, введите В.</b> aaaahm <b>Введите количество блоков,</b> 2</p>	<p><b>РЕШЕНИЕ:</b> A является циклическим сдвигом B. Индекс начала строки B в A: 2</p>
<p><b>Пожалуйста, введите А.</b> ancdefghijkl <b>Пожалуйста, введите В.</b> labcdedfghijk <b>Введите количество блоков,</b> 1</p>	<p><b>РЕШЕНИЕ:</b> A не является циклическим сдвигом B. Индекс начала строки B в A: -1</p>
<p><b>Пожалуйста, введите А.</b> abcdefghijklmn <b>Пожалуйста, введите В.</b> cdefghijklmnab <b>Введите количество блоков,</b> 2</p>	<p><b>РЕШЕНИЕ:</b> A является циклическим сдвигом B. Индекс начала строки B в A: 2</p>
<p><b>Пожалуйста, введите А.</b> aaaaaaa <b>Пожалуйста, введите В.</b> bbbbbbb <b>Введите количество блоков,</b> 2</p>	<p><b>РЕШЕНИЕ:</b> A не является циклическим сдвигом B. Индекс начала строки B в A: -1</p>
<p><b>Пожалуйста, введите А.</b> anastasia <b>Пожалуйста, введите В.</b> anastasiia</p>	<p><b>Введены два эквивалентных текста</b> 0</p>

## **Вывод.**

В результате выполнения лабораторной работы была написана программа, решающая задачу поиска в строке с помощью алгоритма Кнута-Морриса-Пратта. Написанная на языке программирования C++ программа находит все возможные вхождения шаблона в текст, а также определяется является ли одна строка циклическим сдвигом другой, кроме того, была произведена подготовка к распараллеливанию.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД АЛГОРИТМА КНУТА-МОРРИСА-ПРАТТА

```
#include <iostream>
#include <string>
#include <vector>
#include <cmath>

class KMP{ //класс,
реализующий КМП
private:

    int sizeTemp1; //длина
шаблона
    int sizeText; //длина текста
    int task; //0 - поиск
вхождений, 1 - сдвиг?
    int quantityBlock; //кол-во
блоков для разделения текста
    int sizeBlock; //размер блока
    int sizeBlockSk; //максимальный
размер обрабатываемого блока

    std::string templP; //шаблон
    std::string textT; //текст

    std::vector <int> prefixFunc; //префикс-
функция
    std::vector <int> indexEntry; //для
результата
    std::vector <std::string> block; //для хранения
блоков

    void setTemplP(std::string P){ //сеттеры
        templP = P;
    }

    void setTextT(std::string T){
        if(!task) textT = T;
        else textT = T + T; //для
определения сдвига
    }

    void setQuantBlock(int quantBl){
        quantityBlock = quantBl;
    }

    int checkFirst(bool flag){ //для
проверки входных данных при поиске вхождений
```

```

        if(!sizeTempl || !sizeText) { // размер
равен 0 - неверно
            if(flag) std::cout << "Ошибка: Введены не все исходные
данные\n";
            return -1;
        }

        if(sizeTempl > sizeText) { // шаблон
больше текста - неверно
            if(flag) std::cout << "Ошибка: Длина шаблона > Длины
текста\n";
            return -1;
        }

        if(sizeTempl == sizeText && templP == textT) { // один и
тот же текст - ответ 0
            if(flag) std::cout << "Введены два эквивалентных
текста\n";
            return 0;
        }

        return 1; //иначе
запускаем алгоритм
    }

int checkSecond(std::string T, std::string P, bool flag) {
//для проверки входных данных при определении сдвига
    if(!T.length() || !P.length()) {
// размер равен 0 - неверно
        if(flag) std::cout << "Ошибка: Введены не все исходные
данные\n";
        return -1;
    }

    if(P.length() != T.length()) {
// шаблон больше текста - неверно
        if(flag) std::cout << "Ошибка: Длина А должна быть
равна длине B\n";
        return -1;
    }

    if(P.length() == T.length() && P == T) {
// один и тот же текст - ответ 0
        if(flag) std::cout << "Введены два эквивалентных
текста\n";
        return 0;
    }

    return 1;
//иначе запускаем алгоритм
}

bool find(int elem) {

```

```

        for(int i = 0; i < indexEntry.size(); i++){
            if(indexEntry[i] == elem) return true;
        }
        return false;
    }

    void trainingText(){
//разрезание исходного текста на блоки
        std::cout << "\nЭтап 1 - Разделение текста на k = " <<
quantityBlock << " блоков\n";
        std::cout << "Исходный текст: ";
        printText();
        std::cout << "Количество блоков k = " << quantityBlock;
        std::cout << "\n";

        block.resize(quantityBlock);
        sizeBlock = floor(sizeText / quantityBlock);
//размер блока
        if(sizeTemp != 1) sizeBlockSk = sizeBlock + (sizeTemp -
1); //максимальный размер сканируемого блока
        else sizeBlockSk = sizeBlock + 2;

        std::string temp;
        int index;

        std::cout << "Максимальный размер блока: " << sizeBlockSk
<< "\n";

        for(int i = 0; i < quantityBlock; i++){
//разрезание на блоки
            index = sizeBlock * i;

            for(int j = index; j < sizeBlockSk + index; j++) {
                if(j < sizeText)
//отрезаем от исходного текста необходимый блок
                    temp += textT[j];
                else
                    break;
//последний блок может быть меньше максимального размера
            }

            block[i] = temp;
//кладем в вектор блоков
            temp.clear();

            std::cout << "Блок " << i + 1 << ":" " ;
            printBlock(i);
            std::cout << "\n";
        }
        std::cout << "\n";
    }
}

```

```

    void createPrefixFunc() {
        // формирование массива префикс-функции
        std::cout << "Этап 2 - Формирование массива префикс-
функции\n";
        std::cout << "Обрабатываемый шаблон: ";
        printPattern();

        int j = 0;
        // j, i - указывают на рассматриваемые символы
        int i = 1;
        prefixFunc.push_back(0);
        // prefixFunc[0] = 0 - всегда

        while(i < sizeTempl) {
            // пока не просмотрели весь шаблон
            std::cout <<
            ".....\n";
            std::cout << "Обработка: шаблон[" << j << "] = " <<
            templP[j];
            std::cout << " и шаблон[" << i << "] = " << templP[i]
            << "\n";

            if(templP[i] == templP[j]) {
                //если символ повторяется
                std::cout << "Символы совпадают!\n";
                std::cout << "prefixFunc[" << i << "] = " << j + 1
                << "\n";

                prefixFunc[i] = j + 1;
                // фиксируем значение в префикс-функции
                i++;
                // и сдвигаемся
                j++;
            }
            else{
                std::cout << "Символы не совпадают!\n";
                if(!j) {
                    // j == 0
                    std::cout << "prefixFunc[" << i << "] = 0\n";

                    prefixFunc[i] = 0;
                    i++;
                }
                else{
                    // если j не указывает на начало суффикса
                    std::cout << "j = prefixFunc[" << j - 1 << "]"
                    = " << prefixFunc[j-1] << "\n";

                    j = prefixFunc[j - 1];
                    // присваеваем значения префикс-функции предыдущего
                    // символа на который указывает j
                }
            }
        }
    }

```

```

        }
    }

    std::cout << "\nПолученная префикс-функция:\n";
    printPrefixFunc();
}

void patternSearch(std::string block, int index) {
// поиск вхождений шаблона в текст

// Т - текст Р - шаблон
    int indexT = 0;
// стартуем из начала текста и шаблона
    int indexP = 0;

    std::cout << "\nШаблон :";
    printPattern();
    std::cout << "Блок: ";
    printBlock(index);

    while(indexT != block.length()) {
// пока не просмотрели весь текст
        std::cout << "\n\n";
        std::cout << "Символы для сравнения: шаблон[" <<
indexP << "] = " << templP[indexP];
        std::cout << " и блок[" << indexT << "] = " <<
block[indexT];
        std::cout << "\n";

        if(templP[indexP] == block[indexT]) {
// если нашли совпадение
            std::cout << "Символы совпадают!\n";
            indexT++;
            indexP++;
            if(indexP == sizeTempl) {
//не дошли ли до конца шаблона?
                std::cout << "!!!Нашли вхождение шаблона в
текст: ";
                std::cout << indexT+(sizeBlock * index) -
(sizeTempl);
                std::cout << "\n\n";
                if(find(indexT+(sizeBlock * index) -
(sizeTempl))) std::cout << "Вхождение уже записано\n";
                else
                    indexEntry.push_back(indexT+(sizeBlock *
index) - (sizeTempl)); //фиксируем индекс вхождения найденного
шаблона
            }
//учитываем, что индексация в блоках не соответствует
        }
//индексации в исходном тексте
    else{

```

```

        std::cout << "Символы не совпадают! \n";

        if (!indexP)
//indexP == 0
            indexT++;
//сдвигаемся по тексту
        else
            indexP = prefixFunc[indexP - 1];
//перемещаемся на элемент с индексом = префикс-функции предыдущего
элемента
    }
}

void alghorithmKMP () {
//для вызова необходимых методов в строгом порядке
    trainingText();
//1 этап - разрезание текста на блоки
    prefixFunc.resize(templP.length() - 1);
//для префикс-функции
    createPrefixFunc();
//2 этап - формирования массива префикс-функции

    std::cout << "Этап 3 - Поиск вхождений шаблона в текст";

    for(int i = 0; i < quantityBlock; i++) {
        std::cout << "\nОбрабатываемый блок: " << i + 1 << "
";
        printBlock(i);
        patternSearch(block[i], i);
//3 этап - поиск вхождений шаблона в текст
    }

    std::cout << "\n";
    std::cout << "\nРЕШЕНИЕ:\n";
    printAnswer();
//печатать результата
}

public:
    void readData(int task_) {
        task = task_;
//номер задания
        std::string P;
//шаблон
        std::string T;
//текст

        if (!task) {
//поиск вхождений
            std::cout << "Пожалуйста, введите шаблон.\n";

```

```

        std::cin >> P;
        std::cout << "Пожалуйста, введите текст.\n";
        std::cin >> T;
    }
    else{
//определение сдвига
        std::cout << "Пожалуйста, введите А.\n";
        std::cin >> T;
        std::cout << "Пожалуйста, введите В.\n";
        std::cin >> P;

        if(checkSecond(T, P, false) != 1){
//проверка частных случаев
            std::cout << checkSecond(T, P, true);
            std::cout << "\n";
            exit(0);
        }
    }

    setTemplP(P);
//инкапсуляция
    setTextT(T);

    sizeTempl = (int)templP.length();
//для удобства
    sizeText = (int)textT.length();

if(!task && checkFirst(false) != 1){
//проверка частных случаев
    std::cout << checkFirst(true);
    std::cout << "\n";
    exit(0);
}

int quantBl;
std::cout << "Введите количество блоков, на которое хотите
разделить текст.\n";
std::cin >> quantBl;
int flag = 0;

while(!flag){
//проверка кол-ва блоков
    if(quantBl <= 0 || (sizeTempl > 1 && quantBl >
(sizeText / (sizeTempl - 1)))
        || (sizeTempl == 1 && quantBl > sizeText )) {
        std::cout << "Некорректное количество
блоков.\n";
        std::cout << "Попробуйте снова.\n";
        std::cin >> quantBl;
    }
    else{
        flag = 1;
    }
}

```

```

        }
    }

    setQuantBlock(quantBl);

    alghorithmKMP();
//вызов алгоритма
}

void printText() {
    for(int i = 0; i < textT.length(); i++) {
        std::cout << textT[i];
    }std::cout << "\n";
}

void printBlock(int index) {
    std::cout << block[index];
}

void printPattern() {
    for(int i = 0; i < templP.length(); i++) {
        std::cout << templP[i];
    }std::cout << "\n";
}

void printPrefixFunc() {
    for(int i = 0; i < prefixFunc.size(); i++) {
        std::cout <<"Шаблон[" << i << "] = " << templP[i] << "
--> prefixFunc[" << i << "] = " << prefixFunc[i] << "\n";
        }std::cout << "\n";
    }
}

void printAnswer() {
//результат
    if(!task) {
//поиск вхождений
        std::cout << "Вхождения шаблона в текст ";
        if(indexEntry.empty())
            std::cout << "не найдены: -1\n" ;
        else{
            std::cout << "найдены: " ;
            for(int k = 0; k < indexEntry.size(); k++) {

                if(indexEntry.size() - 1 != k)
                    std::cout << indexEntry[k] << ",";
                else
                    std::cout << indexEntry[k] << "\n";
            }
        }
    }
}

```

```

        else{
    //выявление сдвига

        if(indexEntry.empty())
            std::cout << "A не является циклическим сдвигом B.
Индекс начала строки B в A: -1\n";

        else{
            std::cout << "A является циклическим сдвигом B.
Индекс начала строки B в A: ";
            std::cout << indexEntry[0];
            std::cout << "\n";
        }
    }
}

;

int main(int argc, const char * argv[]) {
    setlocale(LC_ALL, "rus");
    char solution = '0';
    int exit = 0;

    std::cout << "Введите a, если вы хотите вывести все вхождения
шаблона P в текст T.\n";
    std::cout << "Введите b, если вы хотите определить является ли
A циклическим сдвигом в B.\n";
    std::cin >> solution;

    do{
        switch(solution){
            case 'a':
                exit = 0;
                break;

            case 'b':
                exit = 0;
                break;

            default:
                std::cout << "Ошибка: некорректная команда -
попробуйте снова.\n";
                std::cin >> solution;
                exit = 1;
        }
    }while(exit == 1);

    KMP start;                                //класс,
реализующий КМП
    start.readData((int)solution - 97);
    return 0;
}

```

