

**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**

Студент гр. 9383

\_\_\_\_\_

Арутюнян С.Н.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2021

## **Цель работы.**

Изучить алгоритм Кнута-Морриса-Пратта и применить его на практике.

## **Основные теоретические положения.**

Сначала введем понятие **префикс-функции**.

Префикс-функция строки  $S$  — это функция, возвращающая массив  $P$ ,  $i$ -й элемент которого равен наибольшей длине наибольшего собственного суффикса подстроки  $S[0...i]$ , совпадающего с ее префиксом.

Пусть  $T$  — шаблон, а  $S$  — исходная строка. Нам нужно определить индексы начала строки  $T$  в строке  $S$ . Мы можем сделать это следующим образом:

1. Построим массив префикс-функции строки  $T + \text{«\#»} + S$ .
2. Пройдемся по значениям этого массива, начиная с  $(|T|+2)$ -го элемента.
3. Если значение текущего элемента этого массива равно длине строки  $T$ , то индекс этого элемента — искомый и мы можем добавить его в ответ.

Этот алгоритм называется **алгоритмом Кнута-Морриса-Пратта**.

## **Задание.**

Реализовать алгоритм Кнута-Морриса-Пратта и решить две задачи:

1. С помощью заданных шаблона  $P$  и строки  $T$  найти все вхождения  $P$  в  $T$ . Если таких вхождений нет, вывести -1.
2. Для двух заданных строк  $A$  и  $B$  определить, является ли  $A$  циклическим сдвигом  $B$ . Если нет, вывести -1.

## **Входные данные:**

1. Шаблон  $P$  ( $|P| \leq 15000$ ) и строка  $T$  ( $|T| \leq 5000000$ ).
2. Строки  $A$  и  $B$  ( $|A|, |B| \leq 5000000$ ).

### **Ход работы:**

1. Была разработана функция PrefixFunction. Алгоритм ее работы таков:

- 1) Инициализируем prefix\_array размера  $|S|$  нулями.
- 2) Проходимся по индексам от 1 до  $|S|$ .
- 3) Определяем  $j = \text{prefix\_array}[i-1]$ .
- 4) Пока  $j > 0$  и  $S[i]$  не равно  $S[j]$ , инициализируем  $j = \text{prefix\_array}[j-1]$ .
- 5) Если  $S[i]$  равно  $S[j]$ , увеличиваем  $j$  на 1.
- 6) Определяем  $\text{prefix\_array}[i] = j$ .

2. Была реализована функция KnutMorrisPratt, которая делает в точности все то, что описано в основных теоретических положениях.

3. Была реализована функция CycleShiftDetect, работающая так же, как и KnutMorrisPratt с некоторыми изменениями: конкатенируются не  $S$  и  $T$ , а  $S+S$  и  $T$ .

4. Для экономии памяти и ускорения работы программы, был разработан класс StringWrapper. Он представляет собой обертку над сконкатенированными строками и нужен для того, чтобы не тратить слишком много времени и памяти для «склеивания» произвольного количества строк.

## Описание функций и структур данных:

1. `inline std::vector<int> PrefixFunction(const StringWrapper& s)` – функция, вычисляющая префиксный массив.

2. `inline std::vector<int> KnutMorrisPratt(const std::string& source, const std::string& substring)` – непосредственно алгоритм Кнута-Морриса-Пратта.

3. `inline int CycleShiftDetect(const std::string& source, const std::string& substring)` – возвращает первый индекс вхождения циклического сдвига.

4. Класс `StringWrapper`. Этот класс представляет собой обертку над `std::string`, позволяющий выполнить конкатенацию без копирования строк. Он имеет следующие методы:

1) `StringWrapper(std::initializer_list<MiniWrapper>)` - конструктор от списка объектов `MiniWrapper`.

2) `StringWrapper(const std::string& str)` – конструктор от `std::string`.

3) `std::size_t size() const` – возвращает количество символов во всех строках.

4) `char operator[](std::size_T index) const` – возвращает символ по индексу. Если индекс выходит за границы, бросается исключение `std::out_of_range`.

5. Структура `MiniWrapper`. Нужна для упрощенной замены `std::string_view`, т.к. `Stepik` не поддерживает C++17 и мне пришлось создавать велосипед.

## Примеры работы программы

```
ab
abab
Префиксный массив:
0 0 0 1 2 1 2
Ответ: 0 2
```

Рисунок 1. Пример работы программы (1)

```
pyat
postavte pyat v zachetku
Префиксный массив:
0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 2 3 4 0 0 0 0 0 0 0 0 0 0 0
Ответ: 9
```

Рисунок 2. Пример работы программы (2)

```
lol
privet privet try to lol find it
Префиксный массив:
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 0 1 0 0 0 0 0 0 0 0
Ответ: -1
```

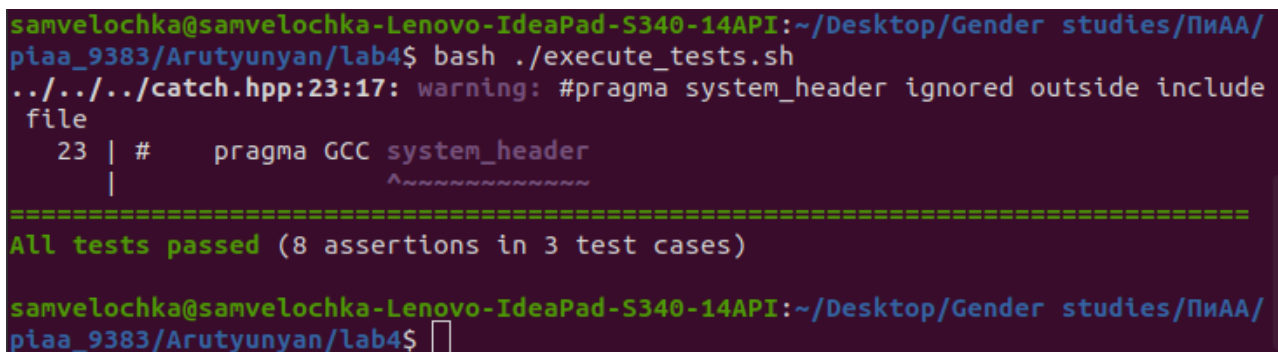
Рисунок 3. Пример работы программы (3)

```
ehalibeniki
ikiehaliben
Префиксный массив:
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 2 3 4 5 6 7 8 9 10 11 1 2 3
Ответ: 8
```

Рисунок 4. Пример работы программы для задания 2

## Тесты

1. **String concatenation tests** — тестирование правильности работы классов ConcatenatingStringsWrapper и StringWrapper.
2. **Knut-Morris-Pratt test** — тестирование правильности работы реализованного алгоритма.
3. **Cycle shift test** — тестирование правильности работы обнаружения циклического сдвига.



```
samvelochka@samvelochka-Lenovo-IdeaPad-S340-14API:~/Desktop/Gender studies/ПиАА/piaa_9383/Arutyunyan/lab4$ bash ./execute_tests.sh
../.././././catch.hpp:23:17: warning: #pragma system_header ignored outside include file
   23 | #    pragma GCC system_header
      |
=====
All tests passed (8 assertions in 3 test cases)

samvelochka@samvelochka-Lenovo-IdeaPad-S340-14API:~/Desktop/Gender studies/ПиАА/piaa_9383/Arutyunyan/lab4$
```

Рисунок 5. Успешное прохождение всех тестов

## **Выводы.**

В выполненной лабораторной работе был изучен и применен на практике алгоритм Кнута-Морриса-Пратта. Также, я нашел способ оптимально конкатенировать произвольное количество строк, чтобы не тратить на наивную конкатенацию слишком много времени и памяти.

## ПРИЛОЖЕНИЕ А

### MAIN.CPP

```
#INCLUDE <Iostream>

#include <Algorithm>

#include "KnutMorrisPratt.hpp"

int main() {
    std::string source, substring;

    std::getline(std::cin, substring);
    std::getline(std::cin, source);

    auto answer = KnutMorrisPratt(source, substring);
    std::cout << "Отбет: ";
    for (int v : answer) {
        std::cout << v << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

### STRINGWRAPPER.HPP

```
#pragma once

struct StringWrapper {
    const char* data;
    std::size_t size_;

    StringWrapper(const std::string& str)
        : data(str.data()), size_(str.size()) {}

    StringWrapper(const char* c_str, std::size_t size)
        : data(c_str), size_(size) {}
}
```



```

STD::SIZE_T SIZE() CONST {
    RETURN SIZE_;
}

CHAR OPERATOR[] (STD::SIZE_T INDEX) CONST {
    RETURN DATA[INDEX];
}
};

CLASS CONCATENATEDSTRINGSWRAPPER {
PUBLIC:
    CONCATENATEDSTRINGSWRAPPER() = DEFAULT;

    CONCATENATEDSTRINGSWRAPPER(STD::INITIALIZER_LIST<STRINGWRAPPER> INIT_STRINGS)
{
    FOR (CONST AUTO& WRAP : INIT_STRINGS) {
        ALL_SIZE += WRAP.SIZE();
        STRINGS.PUSH_BACK(WRAP);
    }
}

    CONCATENATEDSTRINGSWRAPPER(CONST STRINGWRAPPER& INIT_STRING) {
        STRINGS.PUSH_BACK(INIT_STRING);
        ALL_SIZE += INIT_STRING.SIZE();
    }

    CONCATENATEDSTRINGSWRAPPER(CONST STD::STRING& STR) {
        STRINGS.EMPLACE_BACK(STR.DATA(), STR.SIZE());
        ALL_SIZE += STR.SIZE();
    }

STD::SIZE_T SIZE() CONST {
    RETURN ALL_SIZE;
}

```

```
CHAR OPERATOR[] (STD::SIZE_T INDEX) CONST {
```

```
    STD::SIZE_T CURRENT_SIZE = 0;
```

```
    FOR (CONST AUTO& WRAP : STRINGS) {
```

```
        IF (CURRENT_SIZE + WRAP.SIZE() > INDEX)
```

```
            RETURN WRAP[INDEX - CURRENT_SIZE];
```

```
        CURRENT_SIZE += WRAP.SIZE();
```

```
    }
```

```
    THROW STD::OUT_OF_RANGE("INDEX IS OUT OF RANGE!");
```

```
}
```

```
CONCATENATEDSTRINGSWRAPPER OPERATOR+(CONST CONCATENATEDSTRINGSWRAPPER&  
INIT_STRING) CONST {
```

```
    CONCATENATEDSTRINGSWRAPPER UPDATED;
```

```
    UPDATED.ALL_SIZE += ALL_SIZE + INIT_STRING.SIZE();
```

```
    UPDATED.STRINGS.INSERT(UPDATED.STRINGS.END(), STRINGS.BEGIN(), STRINGS.END());
```

```
        UPDATED.STRINGS.INSERT(UPDATED.STRINGS.END(), INIT_STRING.STRINGS.BEGIN(),  
INIT_STRING.STRINGS.END());
```

```
    RETURN UPDATED;
```

```
}
```

```
CONCATENATEDSTRINGSWRAPPER OPERATOR+(CONST STRINGWRAPPER& INIT_STRING)  
CONST {
```

```
    CONCATENATEDSTRINGSWRAPPER UPDATED;
```

```
    UPDATED.ALL_SIZE += ALL_SIZE + INIT_STRING.SIZE();
```

```
    UPDATED.STRINGS.INSERT(UPDATED.STRINGS.END(), STRINGS.BEGIN(), STRINGS.END());
```

```
    UPDATED.STRINGS.PUSH_BACK(INIT_STRING);
```

```
    RETURN UPDATED;
```

```
}
```

```
CONCATENATEDSTRINGSWRAPPER OPERATOR+(CONST STD::STRING& INIT_STRING) CONST {
```

```
    CONCATENATEDSTRINGSWRAPPER UPDATED;
```

```
    UPDATED.ALL_SIZE += ALL_SIZE + INIT_STRING.SIZE();
```

```
    UPDATED.STRINGS.INSERT(UPDATED.STRINGS.END(), STRINGS.BEGIN(), STRINGS.END());
```

```

    UPDATED.STRINGS.EMPLACE_BACK(INIT_STRING);
    RETURN UPDATED;
}

```

```

STD::STRING MAKESTRING() CONST {
    STD::STRING RESULT;

    FOR (CONST AUTO& STRING_WRAPPER : STRINGS) {
        RESULT.APPEND(STRING_WRAPPER.DATA, STRING_WRAPPER.SIZE_);
    }

    RETURN RESULT;
}

```

```

PRIVATE:
    STD::VECTOR<STRING_WRAPPER> STRINGS;
    STD::SIZE_T ALL_SIZE = 0;
};

```

# **KNUTMORRISPRATT.HPP**

```

#pragma ONCE

```

```

#include <VECTOR>
#include <STRING>

```

```

#include "STRINGWRAPPER.HPP"

```

```

STATIC INLINE CONST BOOL DEBUG = FALSE;

```

```

INLINE STD::VECTOR<INT> PREFIXFUNCTION(CONST CONCATENATEDSTRINGSWRAPPER& S) {
    STD::VECTOR<INT> PREFIX_ARRAY(S.SIZE());
}

```

```

FOR (INT I = 1; I < S.SIZE(); ++I) {
    INT J = PREFIX_ARRAY[I-1];
    WHILE (J > 0 && s[I] != s[J])
        J = PREFIX_ARRAY[J-1];

    IF (s[I] == s[J])
        ++J;
    PREFIX_ARRAY[I] = J;
}

RETURN PREFIX_ARRAY;
}

INLINE STD::VECTOR<INT> KNUTMORRISPRATT(CONST CONCATENATEDSTRINGSWRAPPER&
SOURCE,
                                     CONST CONCATENATEDSTRINGSWRAPPER& SUBSTRING) {
    STD::VECTOR<INT> START_INDEXES;

    AUTO PREFIX = PREFIXFUNCTION(SUBSTRING + STRINGWRAPPER{"#", 1} + SOURCE);

    IF (DEBUG) {
        STD::COUT << "ПРЕФИКСНЫЙ МАССИВ:" << STD::ENDL;
        FOR (INT V : PREFIX) {
            STD::COUT << V << " ";
        }
        STD::COUT << STD::ENDL;
    }

    FOR (INT I = 0; I < SOURCE.SIZE(); ++I) {
        IF (PREFIX[SUBSTRING.SIZE() + 1 + I] == SUBSTRING.SIZE())
            START_INDEXES.PUSH_BACK(I - SUBSTRING.SIZE() + 1);
    }

    IF (START_INDEXES.EMPTY())

```

```

        START_INDEXES.PUSH_BACK(-1);
    RETURN START_INDEXES;
}

inline int CycleShiftDetect(const std::string& source,
                           const std::string& substring) {
    return source.size() == substring.size()
        ? KnutMorrisPratt(ConcatenatedStringsWrapper(substring) + substring,
source)[0]
        : -1;
}

```