

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: Кнут-Моррис-Пратт

Студент гр. 9383

Гладких А.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Применить на практике знания о построение алгоритма Кнута-Морриса-Пратта. Реализовать алгоритм Кнута-Морриса-Пратта для поиска всех подстрок по заданному шаблону. Реализовать алгоритм проверки, является ли одна строка циклическим сдвигом другой.

Основные теоретические положения.

Пусть $L = c_0, c_1, \dots, c_{n-1}$ — строка длины n .

Любая строка $S = c_i, \dots, c_j$, где $0 \leq i \leq j \leq n$, является подстрокой L длины $j - i + 1$.

Если $i=0$, то S называется префиксом L длины $j + 1$. Если $j = n - 1$, то S — суффикс L длины $j - i + 1$.

Префикс-функция — функция, которая для заданной строки S с длиной n вычисляет массив чисел $\pi[0], \dots, \pi[n - 1]$, где $\pi[i]$ определяется согласно следующим правилам: это такая наибольшая длина наибольшего собственного суффикса подстроки $S[0...i]$, совпадающего с её префиксом (собственный суффикс — значит не совпадающий со всей строкой). Изначально значение $\pi[0]$ полагается равным 0.

Алгоритм Кнута-Морриса-Пратта (КМП-алгоритм) — эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объёма входных данных.

В простой реализации алгоритм состоит в следующем: дан шаблон P и текст T , требуется найти и вывести позиции всех вхождений шаблона P в текст T . Обозначим для удобства через n длину шаблона P , а через m — длину текста T . Образует строку $P + \# + T$, где символ $\#$ — это разделитель, который не принадлежит исходному алфавиту. Посчитаем для этой строки префикс-функцию и рассмотрим её значения. В месте, где достигается равенство $\pi[i] = n$, оканчивается искомое вхождение шаблона P .

Задание.

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.

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 - A

Вторая строка – B

Выход:

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

Ход работы:

1. Произведён анализ задания.
2. Был реализован алгоритм вычисления префикс-функции:
 1. Сперва создается массив, в который в дальнейшем будут записаны значения префикс-функции. Размер массива равен длине заданной строки S , для которой считается префикс-функция. Элемент $\pi[0]$ равен 0.
 2. Для подсчёта текущего значения $\pi[i]$ заводится переменная j , обозначающая длину текущего рассматриваемого образца. Изначально $j = \pi[i-1]$.

3. Сравниваются символы $S[j]$ и $S[i]$. Если они совпадают — то $\pi[i] = j+1$ и алгоритм переходит к следующему индексу $i+1$. Если же символы отличаются, то j становится равна $\pi[j-1]$, и данный шаг повторяется с начала.
4. Если алгоритм дошел до $j=0$ и так и не были найдены совпадения, то $\pi[i] = 0$ и осуществляется переход к следующему индексу $i+1$.
3. Был реализован алгоритм Кнута-Морриса-Пратта:
 1. Сперва считается префикс-функция для строки вида $P + \#$, где P — шаблон строки, которую необходимо найти в тексте T .
 2. После этого с помощью цикла `for` программа проходит по тексту, сравнивая символы. При этом сдвиг происходит не на один символ, как при наивном сравнении, а на значение префикс-функции для данного символа.
 3. Постепенно проходясь по тексту программа записывает в структуру `vector` из стандартной библиотеки языка $C++$ индексы, в которых текущая рассматриваемая подстрока текста совпадает с шаблонной строкой.
 4. Сложность алгоритма вычисления префикс-функции как по памяти, так и по времени линейна — $O(|S|)$, где $|S|$ — длина строки, для которой нужно вычислить префикс-функцию.
 5. Сложность алгоритма Кнута-Морриса-Пратта по памяти — $O(|P|)$, где $|P|$ — длина строки шаблона, для которой вычисляется префикс-функция. По времени же сложность можно оценить как $O(|P| + |T|)$, где $|P|$ — длина строки шаблона, а $|T|$ — длина текста, в котором ведется поиск. Фактически сложность состоит из построения префикс-функции для шаблона и дальнейшего прохода по всем символам текста.
 6. Задача 1 была решена простым применением алгоритма Кнута-Морриса-Пратта.

7. Задача 2 была решена с помощью алгоритма Кнута-Морриса-Пратта с удвоенной строкой текста T, так как в такой строке точно встретится шаблон P, если текст действительно является циклическим сдвигом шаблона.
8. Для удобства выбора заданий, были разработаны ключи командной строки для запуска соответствующих заданий: ключ «-kmp» запускает выполнение задачи 1, ключ «-cycle» запускает выполнение задачи 2.
9. Были написаны тесты с использованием библиотеки Catch2 для функций программы:
 1. Была протестирована функция считывания строк.
 2. Была протестирована корректная работа функции вычисления префикс-функции для заданной строки — как для непустой, так и для пустой.
 3. Была протестирована функция, реализующая алгоритм Кнута-Морриса-Пратта. Функция была протестирована на разных входных данных, в том числе и тех, когда шаблон или текст являются пустыми строками.
 4. Была протестирована функция поиска циклического сдвига. Функция была протестирована на разных входных данных, в том числе и тех, когда шаблон или текст являются пустыми строками.
 5. Тесты, описанные в данном разделе, представлены в разделе Тестирование функций.
10. Код разработанной программы расположен в Приложении А.

Описание функций и структур данных.

1. Функция *prefix_function()* - вычисляет значение префикс-функции для заданной строки. Возвращает массив значений префикс-функции.
2. Функция *kmp()* - реализует алгоритм Кнута-Морриса-Пратта. Возвращает массив с индексами всех вхождений шаблона в текст.

3. Функция *check_cycle()* - проверяет, является ли одна строка циклическим сдвигом другой. Возвращает индекс начала вхождения строки, если строка — циклический сдвиг, и -1 в противном случае.
4. Функция *read_strings()* считывает две строки из заданного потока. Возвращает переменную типа bool, которая сигнализирует о корректном или некорректном вводе.
5. Функция *print_vector()* выводит массив индексов в соответствие с требованиями в задании 1.
6. Функция *main()* - функция, в которой происходит считывание входных данных и запуск выполнения алгоритма.

Примеры работы программы.

Таблица 1 – Пример работы программы в задании №1

№ п/п	Входные данные	Выходные данные
1.	ab abab	0,2
2.	needle stackstackneedlestaneedle	10,19
3.	ab abababababab	0,2,4,6,8,10

Таблица 2 – Пример работы программы в задании №2

№ п/п	Входные данные	Выходные данные
1.	defabc abcdef	3
2.	abaa baaa	1
3.	aaa aaa	0

Иллюстрация работы программы.

```
ab
abab
0,2
```

Рисунок 1 - Пример работы программы в задании №1 на входных данных №1

```
needle
stackstackneedlestaneedle
10,19
```

Рисунок 2 - Пример работы программы в задании №1 на входных данных №2

```
ab
ababababababab
0,2,4,6,8,10
```

Рисунок 3 - Пример работы программы в задании №1 на входных данных №3

```
defabc
abcdef
3
```

Рисунок 4 - Пример работы программы в задании №2 на входных данных №1

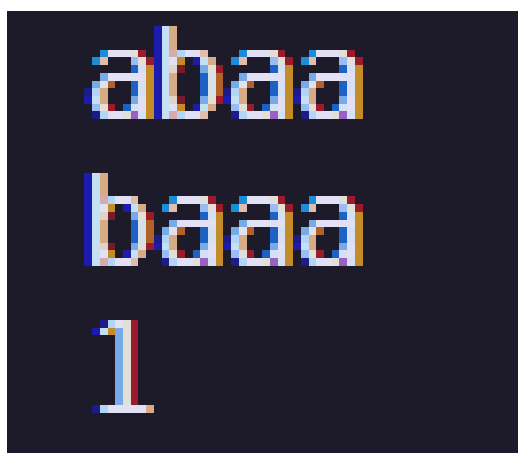


Рисунок 5 - Пример работы программы в задании №2 на входных данных №2

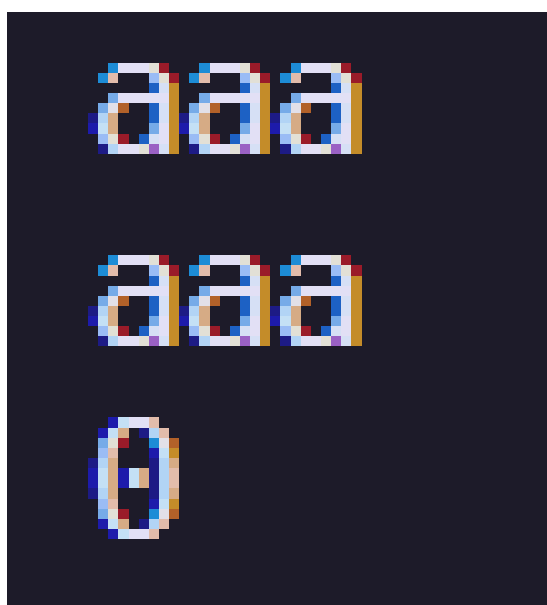


Рисунок 6 - Пример работы программы в задании №2 на входных данных №3

Тестирование функций.

Таблица 3 – Тестирование функций

Название тестируемой функции	Входные данные	Выходные данные
read_strings()	abc abcd	true
read_strings()	def (пустая строка)	false
prefix_function()	efefeftef	[0, 0, 1, 2, 3, 4, 0, 1, 2]
prefix_function()	(пустая строка)	[]
kmp()	ab ababacdab	[0, 2, 7]
kmp()	(пустая строка) abcdef	[-1]
kmp()	abc (пустая строка)	[-1]
check_cycle()	abaa baaa	1
check_cycle()	defabc abcdef	3
check_cycle()	abaaddd baaa	-1
check_cycle()	abaa b	-1
check_cycle()	(пустая строка) abcdef	-1
check_cycle()	abc (пустая строка)	-1

Выводы.

Были применены на практике знания о построении алгоритма Кнута-Морриса-Пратта. Был реализован алгоритм вычисления префикс-функции на языке программирования C++. Был реализован алгоритм Кнута-Морриса-Пратта на языке программирования C++. С помощью реализованных алгоритмов были решены два задания: поиск всех подстрок по заданному шаблону в заданном тексте и проверка, является ли одна строка циклическим сдвигом другой.

ПРИЛОЖЕНИЕ А

ФАЙЛ MAIN.CPP

```
#INCLUDE <Iostream>

#include "KMP.HPP"

INT MAIN(INT ARGV, CHAR** ARGV) {

    INT TASK = -1;

    FOR(INT I = 0; I < ARGV; ++I) {
        IF(STD::STRING(ARGV[I]) == "-KMP") {
            TASK = 1;
            BREAK;
        }

        ELSE IF(STD::STRING(ARGV[I]) == "-CYCLE") {
            TASK = 2;
            BREAK;
        }

        ELSE IF(STD::STRING(ARGV[I]) == "-H" || STD::STRING(ARGV[I]) == "--
HELP") {
            TASK = 0;
            BREAK;
        }
    }

    IF(TASK == -1) {
        STD::COUT << "YOU HAVE TO SPECIALIZE THE TASK: EITHER CHOOSE KMP OR
CYCLE CHECK.\nTo GET MORE INFO USE KEY -H OR --HELP.\n";
        RETURN 0;
    }
    ELSE IF(TASK == 0) {
        STD::COUT << "USE -KMP TO START KNUTH-MORRIS-PRATT ALGORITHM\n";
        STD::COUT << "USE -CYCLE TO CHECK IF A STRING IS A CYCLE SHIFT OF
ANOTHER ONE\n";
        RETURN 0;
    }

    STD::STRING PATTERN, TEMP;

    IF(!READ_STRINGS(PATTERN, TEMP, STD::CIN)) {
        STD::COUT << "YOU'VE ENTERED EMPTY STRING\n";
        RETURN 0;
    }

    IF(TASK == 1) {
        STD::VECTOR<INT> RES = KMP(PATTERN, TEMP, FALSE);
        PRINT_VECTOR(RES);
        RETURN 0;
    }
    ELSE {
        STD::COUT << CHECK_CYCLE(PATTERN, TEMP) << '\n';
        RETURN 0;
    }
}
```

```

    }

    RETURN 0;
}

```

ФАЙЛ КМР.НПП

```

#pragma once

#include <vector>
#include <string>
#include <iostream>

std::vector<int> PREFIX_FUNCTION(const std::string& str);
std::vector<int> KMP(const std::string& patt, const std::string& temp, bool
stop_at_first);
int check_cycle(const std::string& patt, const std::string& temp);

bool read_strings(std::string& patt, std::string& temp, std::istream& in);
void print_vector(const std::vector<int>& vec);

```

ФАЙЛ КМР.CPP

```

#include "kmp.hpp"

std::vector<int> PREFIX_FUNCTION(const std::string& str) {
    int n = str.length();
    if (n == 0) return std::vector<int> (0);

    std::vector<int> prefix_arr(n);

    int j;
    for (int i = 1; i < n; ++i) {
        j = prefix_arr[i - 1];
        while (j > 0 && str[i] != str[j]) j = prefix_arr[j - 1];
        if (str[i] == str[j]) j++;
        prefix_arr[i] = j;
    }
    return prefix_arr;
}

std::vector<int> KMP(const std::string& patt, const std::string& temp, bool
stop_at_first) {
    std::vector<int> answer;
    int patt_len = patt.size();
    int temp_len = temp.size();

    if (patt_len == 0 || temp_len == 0) {
        answer.push_back(-1);
        return answer;
    }

    std::vector<int> p = PREFIX_FUNCTION(patt + "#");
    int j = 0;
    for (int i = 0; i < temp_len; ++i) {

```

```

        WHILE (J > 0 && PATT[J] != TEMP[I]) J = P[J-1];
        IF (PATT[J] == TEMP[I]) J++;
        IF (J == PATT_LEN) {
            ANSWER.PUSH_BACK(I - PATT_LEN + 1);
            IF (STOP_AT_FIRST) BREAK;
        }
    }

    IF (!ANSWER.SIZE()) ANSWER.PUSH_BACK(-1);

    RETURN ANSWER;
}

INT CHECK_CYCLE(CONST STD::STRING& PATT, CONST STD::STRING& TEMP) {
    INT PATT_LEN = PATT.SIZE();
    INT TEMP_LEN = TEMP.SIZE();

    IF (PATT_LEN == 0 || TEMP_LEN == 0) RETURN -1;

    IF (PATT_LEN != TEMP_LEN) RETURN -1;

    STD::VECTOR<INT> RES = KMP(TEMP, PATT + PATT, TRUE);

    RETURN RES[0];
}

BOOL READ_STRINGS(STD::STRING& PATT, STD::STRING& TEMP, STD::ISTREAM& IN) {
    IN >> PATT;
    IN >> TEMP;

    IF (PATT.SIZE() == 0 || TEMP.SIZE() == 0) RETURN FALSE;
    RETURN TRUE;
}

VOID PRINT_VECTOR(CONST STD::VECTOR<INT>& VEC) {
    FOR (INT I = 0; I < VEC.SIZE(); ++I) {
        IF (I == VEC.SIZE() - 1) {
            STD::COUT << VEC[I] << "\n";
        }
        ELSE {
            STD::COUT << VEC[I] << ",";
        }
    }
}

```

ФАЙЛ TEST.CPP

```

#define CATCH_CONFIG_MAIN

#include "../.. /CATCH.HPP"
#include "../.. /SOURCE/KMP.HPP"

#include <SSTREAM>

```

```

TEST_CASE("READ FUNCTION TEST", "[INTERNAL KMP TEST]" ) {
    STD::STRING STR1, STR2;
    STD::STRINGSTREAM INPUT_TEST;
    INPUT_TEST << "ABC\NABCD";

    BOOL READ_RESULT = READ_STRINGS(STR1, STR2, INPUT_TEST);

    REQUIRE(READ_RESULT == TRUE);
    REQUIRE(STR1 == "ABC");
    REQUIRE(STR2 == "ABCD");

    STR1.CLEAR();
    STR2.CLEAR();
    STD::STRINGSTREAM INPUT_TEST2;
    INPUT_TEST2 << "DEF\N";

    READ_RESULT = READ_STRINGS(STR1, STR2, INPUT_TEST2);

    REQUIRE(READ_RESULT == FALSE);
}

TEST_CASE("PREFIX FUNCTION TEST", "[INTERNAL KMP TEST]" ) {
    STD::STRING STR = "EFEFEFTEF";

    STD::VECTOR<INT> RES = PREFIX_FUNCTION(STR);
    STD::VECTOR<INT> CORRECT_ANS {0, 0, 1, 2, 3, 4, 0, 1, 2};

    REQUIRE(RES == CORRECT_ANS);

    STR.CLEAR();
    RES = PREFIX_FUNCTION(STR);

    REQUIRE(RES.SIZE() == 0);
}

TEST_CASE("KNUTH-MORRIS-PRATT ALGORITHM TEST", "[INTERNAL KMP TEST]" ) {
    STD::STRING PATT, TEMP;

    PATT = "AB";
    TEMP = "ABABACDAB";

    STD::VECTOR<INT> RES = KMP(PATT, TEMP, FALSE);
    STD::VECTOR<INT> CORRECT_ANS {0, 2, 7};

    REQUIRE(RES == CORRECT_ANS);

    PATT = "";
    TEMP = "ABCDEF";

    RES = KMP(PATT, TEMP, FALSE);
    CORRECT_ANS = {-1};

    REQUIRE(RES == CORRECT_ANS);

    PATT = "ABC";
    TEMP = "";

```

```

    RES = KMP(PATT, TEMP, FALSE);
    CORRECT_ANS = {-1};

    REQUIRE (RES == CORRECT_ANS);
}

TEST_CASE("CHECK CYCLE FUNCTION TEST", "[INTERNAL KMP TEST]" ) {
    STD::STRING PATT, TEMP;

    PATT = "ABAA";
    TEMP = "BAAA";

    INT RES = CHECK_CYCLE(PATT, TEMP);

    REQUIRE (RES == 1);

    PATT = "DEFABC";
    TEMP = "ABCDEF";

    RES = CHECK_CYCLE(PATT, TEMP);

    REQUIRE (RES == 3);

    PATT = "ABAADDD";
    TEMP = "BAAA";

    RES = CHECK_CYCLE(PATT, TEMP);

    REQUIRE (RES == -1);

    PATT = "ABAA";
    TEMP = "B";

    RES = CHECK_CYCLE(PATT, TEMP);

    REQUIRE (RES == -1);

    PATT = "";
    TEMP = "ABCDEF";

    RES = CHECK_CYCLE(PATT, TEMP);

    REQUIRE (RES == -1);

    PATT = "ABC";
    TEMP = "";

    RES = CHECK_CYCLE(PATT, TEMP);

    REQUIRE (RES == -1);
}

```

ФАЙЛ MAKEFILE

```

FLAGS = -STD=C++17 -WALL -WEXTRA

```

```

BUILD = BUILD
SOURCE = SOURCE
TEST = TEST

$(SHELL MKDIR -P $(BUILD))

ALL: LAB4 RUN_TESTS

LAB4: $(BUILD)/MAIN.O $(BUILD)/KMP.O
    @ECHO "To start enter ./LAB4.\nUse -h or --help for list of commands."
    @G++ $(BUILD)/MAIN.O $(BUILD)/KMP.O -o LAB4 $(FLAGS)

RUN_TESTS: $(BUILD)/TEST.O $(BUILD)/KMP.O
    @ECHO "To run tests enter ./RUN_TESTS"
    @G++ $(BUILD)/TEST.O $(BUILD)/KMP.O -o RUN_TESTS

$(BUILD)/MAIN.O: $(SOURCE)/MAIN.CPP
    @G++ -c $(SOURCE)/MAIN.CPP -o $(BUILD)/MAIN.O

$(BUILD)/KMP.O: $(SOURCE)/KMP.CPP $(SOURCE)/KMP.HPP
    @G++ -c $(SOURCE)/KMP.CPP -o $(BUILD)/KMP.O

$(BUILD)/TEST.O: $(TEST)/TEST.CPP
    @G++ -c $(TEST)/TEST.CPP -o $(BUILD)/TEST.O

CLEAN:
    @RM -rf $(BUILD)/
    @RM -rf *.O LAB4 RUN_TESTS

```