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 $0 \le i \le j \le n$, 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.

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. Образуем 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 вляется 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 π[i] = j+1 и алгоритм переходит к следующему индексу i+1. Если же символы отличаются, то j становится равна π[j-1], и данный шаг повторяется с начала.
4. Если алгоритм дошел до j=0 и так и не были найдены совпадения, то π[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 массиву значений префикс-функции, полученному на прошлом шаге, и записывает в структуру vector из стандартной библиотеки языка C++ индексы, в которых значение префикс-функции равно длине строки P.
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 удвоения строки 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 алгоритма 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.

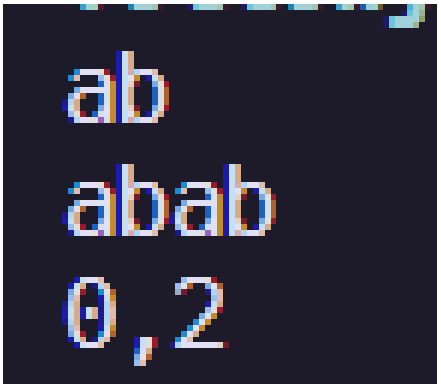

Рисунок 1 - Пример работы программы в задании №1 на входных данных №1

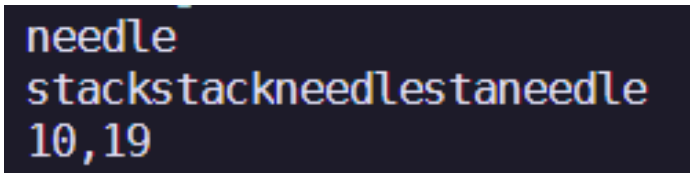

Рисунок 2 - Пример работы программы в задании №1 на входных данных №2

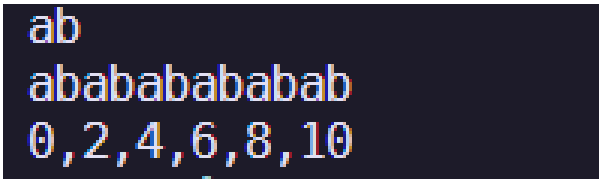

Рисунок 3 - Пример работы программы в задании №1 на входных данных №3

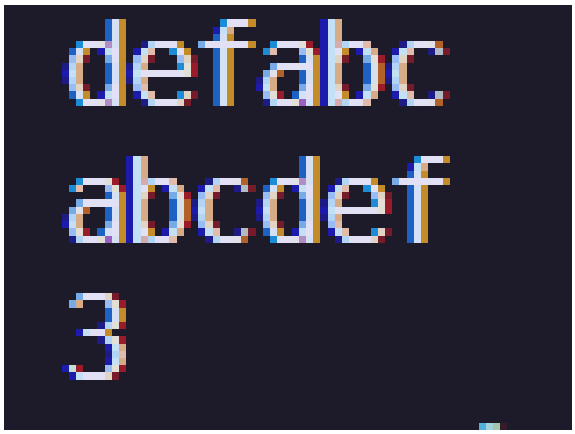

Рисунок 4 - Пример работы программы в задании №2 на входных данных №1

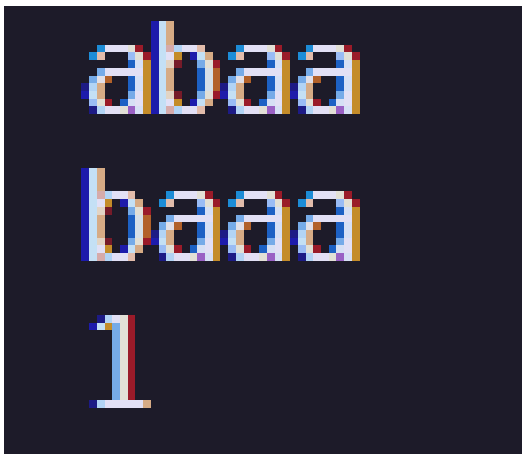

Рисунок 5 - Пример работы программы в задании №2 на входных данных №2

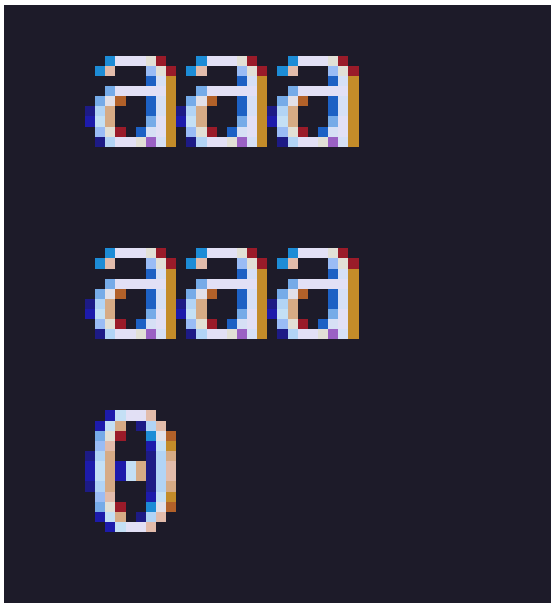

Рисунок 6 - Пример работы программы в задании №2 на входных данных №3

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 построение алгоритма Кнута-Морриса-Пратта. Была 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 argc, char** argv) {

     int task = -1;

     for(int i = 0; i < argc; ++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 KMP.hpp

```
#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 KMP.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
```