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. 9382 _____ Павлов Р.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы

Изучение и реализация алгоритма Кнута-Морриса-Пратта для поиска вхождений шаблона в текст.

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

Sample Input:

```
ab
abab
```

Sample Output:

```
0,2
```

Задание 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.

Sample Input:

```
defabc
abcdef
```

Sample Output:

```
3
```

Описание алгоритма

1. Алгоритм Кнута-Морриса-Пратта.

1) Строится префикс-функция на основании шаблона. Префикс-функция строки – это набор значений, показывающих наибольшую длину суффикса строки, совпадающего с её префиксом (для каждого i-го символа строки префикс и суффикс – это начало от первого символа и конец до i-го символа строки, имеющих одинаковую длину).

2) После того, как построена префикс-функция шаблона, он сравнивается с символами строки. Поскольку префикс-функция базируется на информации о префиксах и суффиксах, сравнение далеко не всегда приходится проводить, сдвигаясь лишь на один символ – сдвиг может быть близок по длине к длине шаблона, а в ином случае осуществляется такой сдвиг, который не позволяет пропустить ни одного вхождения, если такое существует. На текущем символе строки, в которой ищутся вхождения, определяется оптимальный префикс шаблона, совпадающий с суффиксом текущей строки, и сравниваются их активные элементы.

2. Алгоритм проверки двух строк на циклический сдвиг.

1) Сравниваются длины строк, затем, если они равны, алгоритмом КМП ищется первый индекс вхождения второй из них в удвоенную первую. Только в том случае, если вхождение (или несколько вхождений) найдено, первая строка является циклическим сдвигом второй, иначе – не является.

Сложность алгоритма

1. Алгоритм КМП.

Сложность по памяти.

Поскольку для хранения префикс-функции требуется массив, равный длине шаблона, то количество занимаемой памяти линейно, а сложность алгоритма по памяти – O(**m**), где **m** – длина шаблона.

Сложность по времени.

Индекс текущего элемента строки на каждой итерации поиска сдвигается вперёд на одну позицию, т.е. повторно её символы не сравниваются с символами шаблона – префикс-функция позволяет для каждого текущего символа определять, совпадает ли текущее значение шаблона с ним или нет, и сложность алгоритма по времени также линейна – O(**n**), где **n** – длина строки, в которой ищутся вхождения.

2. Алгоритм проверки на циклический сдвиг.

Сложность по памяти.

Сложность по памяти остаётся линейной в случае проверки на сдвиг, т.к. шаблон – вторая строка – имеет свою изначальную длину, и алгоритм выполняется за O(**m**).

Сложность по времени.

Поскольку первая строка удваивается перед поиском, может удвоиться также и время поиска шаблона, но она всё так же остаётся линейной – O(**2n**), т.е. O(**n**).

Описание функций и СД

- **std::vector<int> prefix_func(std::string str) –** вычисление префикс-функции; возвращаемое значение – массив значений префикс-функции; **str** – строка, для которой вычисляется префикс-функция

- **std::vector<int> indVec(const std::string& pattern, const std::string& text) –** поиск вхождений подстроки в строку; возвращаемое значение – набор индексов, начиная с которых в строке воспроизводится подстрока (одно значение -1, если вхождений нет); **pattern** – искомая подстрока, **text** – строка, в которой осуществляется поиск

- **int isCyclicShift(const std::string& A, const std::string& B) –** проверка на то, является ли строка **A** циклическим сдвигом строки **B**;

возвращаемое значение – первый индекс вхождения строки **B** в строку **A** (-1, если не является циклическим сдвигом); **A** - проверяемая строка, **B** – строка соответствия

- **int main() –** ввод исходных данных, вывод результатов работы программы

Тестирование

Результаты тестирования для задания 1 представлены в таблице 1.

Таблица 1

Входные данные	Выходные данные
aba abababbabab	0,2,7
crunch kascrunascrurcrunchasiccrunch	13,23
qwerty odaspmcqwertuyainafiyuh	-1

Результаты тестирования для задания 2 представлены в таблице 2.

Таблица 2

Входные данные	Выходные данные
obaab aboba	3
obama obada	-1
oposoow ooposoow	-1

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

Имя файла: alg_lab4.cpp

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

#define TASK 1
//#define OUTPUT

std::vector<int> prefix_func(std::string str) {
#ifdef OUTPUT
    std::cout << "Построение префикс-функции для строки " << str << "\n";
#endif
    std::vector<int> pi(str.length(), 0);
    int i = 1, j = 0;
    while (i < str.length()) {
#ifdef OUTPUT
        std::cout << "\nСравниваются " << j + 1 << "-й символ (" << str[j]  << ") и " << i +
1 << "-й символ (" << str[i] << ") строки\n";
#endif
        if (str[i] == str[j]) {
#ifdef OUTPUT
            std::cout << "Символы равны, сравнение продолжается на позицию дальше\n";
#endif
            pi[i++] = ++j;
        }
        else if (j != 0) {
#ifdef OUTPUT
            std::cout << "Символы не равны, возврат к префиксу последнего совпавшего символа
(" << str[j - 1] << ")\n";
#endif
            j = pi[j - 1];
        }
        else {
#ifdef OUTPUT
            std::cout << "Символы не равны и не найдено ни одного подходящего ненулевого
префикса, сдвиг на позицию вперёд\n";
#endif
            pi[i++] = 0;
        }
    }
#ifdef OUTPUT
    std::cout << "\nПрефикс-функция построена и имеет следующий вид: ";
    for (auto &el : pi) {
        std::cout << el << " ";
    }
    std::cout << "\n";
#endif
    return pi;
}

std::vector<int> indVec(const std::string& pattern, const std::string& text) {
    std::vector<int> indVec;
    std::vector<int> prefix = prefix_func(pattern);
    int satisfied = pattern.length();

    int k = 0, l = 0;
#ifdef OUTPUT
    std::cout << "\nОсуществляется поиск шаблона " << pattern << " в строке " << text <<
"\n";
```

```
#endif
    while (k < text.length()) {
#ifdef OUTPUT
        std::cout << "\n";
#endif
        if (text[k] == pattern[l]) {
#ifdef OUTPUT
            std::cout << l << "-й символ шаблона (" << pattern[l] << ") и " << k << "-й
символ строки (" << text[k] << ") совпадают\n";
#endif
            k++;
            l++;
            if (l == satisfied) {
#ifdef OUTPUT
                std::cout << "\nНайдено вхождение шаблона в строку, индекс символа шаблона
меняется на предыдущий префикс pi(" << pattern[l - 1] << ") = " << prefix[l - 1] << "\n";
#endif
                l = prefix[l - 1];
                indVec.push_back(k - satisfied);
            }
        }
        else if (l != 0) {
#ifdef OUTPUT
            std::cout << "Символы " << pattern[l] << " и " << text[k] << " не равны, возврат
к префиксу последнего совпавшего символа (" << pattern[l - 1] << ")\n";
#endif
            l = prefix[l - 1];
        }
        else {
#ifdef OUTPUT
            std::cout << "Символы " << pattern[l] << " и " << text[k] << " не равны и не
найдено ни одного подходящего ненулевого префикса, сдвиг на позицию вперёд, сравнение с
начала шаблона\n";
#endif
            k++;
        }

    }
#ifdef OUTPUT
    std::cout << "Поиск закончен, ";
#endif
    if (indVec.empty()) {
#ifdef OUTPUT
        std::cout << "не найдено ни одного вхождения\n";
#endif
        indVec.push_back(-1);
    }
#ifdef OUTPUT
    else {
        std::cout << "индексы вхождения шаблона в строку: ";
        for (auto& el : indVec) {
            std::cout << el << " ";
        }
        std::cout << "\n";
    }
#endif

    return indVec;
}

int isCyclicShift(const std::string& A, const std::string& B) {
#ifdef OUTPUT
    std::cout << "Проверка строки " << A << " на соответствие циклическому сдвигу строки "
<< B << "\n";
#endif
```

```
    if (A.length() != B.length()) {
#ifdef OUTPUT
        std::cout << "Длины сравниваемых строк не равны, циклический сдвиг невозможен.\n";
#endif
        return -1;
    }
    return indVec(B, A + A)[0];
}

int main()
{
    setlocale(LC_ALL, "rus");
    std::string str1 = "", str2 = "";
    std::cin >> str1 >> str2;

    if (TASK == 1) {
        std::vector<int> index = indVec(str1, str2);
        for (int i = 0; i < index.size(); i++) {
            std::cout << index[i];
            if (i < index.size() - 1) {
                std::cout << ",";
            }
        }
    }
    else if (TASK == 2) {
        int cyclic = isCyclicShift(str1, str2);
#ifdef OUTPUT
        if (cyclic > -1) {
            std::cout << str1 << " является циклическим сдвигом " << str2 << "\n";
        }
        else {
            std::cout << str1 << " не является циклическим сдвигом " << str2 << "\n";
        }
#endif
        std::cout << cyclic;
    }
    else {
        std::cout << "Неизвестный номер задания, завершение работы\n";
    }

    std::cout << "\n";
    return 0;
}
```