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-Мориса-Пратта

Студент гр. 1304 Шаврин А.П.

Преподаватель Шевелева А.М.

Санкт-Петербург
2023

Цель работы.

Изучить и реализовать алгоритм Кнута-Морриса-Пратта поиска подстроки в строке, а также изучить и решить задачу определения циклического сдвига.

Задание.

1. 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. Задание 2.

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

Выполнение работы.

1. Сначала был реализован класс Solver, который содержит поля и методы, необходимые для решения обоих задач.

Данный класс имеет следующие приватные поля:

- std::string first_string – первая строка из входных данных.
- std::string second_string – вторая строка из входных данных.

Данный класс имеет следующие приватные методы:

- std::vector<int> prefixFunction(std::string string) – метод считает значение префикс функции и возвращает массив длин максимальных совпадений префиксов и суффиксов. Префикс-функция от строки s равна массиву prefix_indexes, где prefix_indexes[i] обозначает длину максимального префикса строки s[0..i], совпадающего с её суффиксом.

Данный класс имеет следующие публичные методы:

- Solver() – конструктор класса, в котором происходит считывание входных данных.
- std::string KMPAlgorithm() – функция реализующая алгоритм Кнута-Морриса-Пратта, используя трюк. Трюк заключается в том, что в функцию передается не искомая подстрока, а строка вида: искомая подстрока + разделительный символ (‘@’) + строка, которой ищется подстрока. Символ '@' играет роль разделителя, его заведомо нет ни в образце, ни в строке поиска. Префиксная функция массив, в котором максимальные элементы, равны длине образца. Значений больше длины образца не будет из-за символа-разделителя, а значения, равные длине образца могут появиться только в позициях, соотвествующих исходной строке поиска. Склеенная строка имеет длину <длина образца>+<длина строки>, поэтому время расчета оценивается как O(<длина образца>+<длина строки>).
- int cycleShiftAlgorithm() – метод решает задачу о циклическом сдвиге. В связи с ограничением по памяти в данном методе было решено не

склеивать первую строку саму с собой, а зациклить индекс для данной строки. Данное решение позволяет определить индекс начала второй строки в первой, если она является циклическим сдвигом первой строки при этом не используя дополнительную память.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	ab abab	0,2	Тест задания 1
2.	defabc abcdef	3	Тест задания 2

Выводы.

Изучен и реализован алгоритм Кнута-Морриса-Пратта поиска подстроки в строке, а также изучена и решена задачу определения циклического сдвига.

Обе задачи были реализованы в качестве методов одного класса, поскольку имеют схожий смысл.

Алгоритм Кнута-Морриса-Пратта имеет линейное решение, благодаря префикс функции. Время расчета оценивается как O(<длина образца>+<длина строки>, благодаря трюку со склеиванием строк.

Благодаря второй задаче, был выявлен недостаток данного алгоритма при прямом решении – его затраты на память. Однако благодаря зацикливанию индекса первой строки, этот недостаток можно исправить.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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


/**
 * This class solves the problem of finding occurrences of a substring in a string
**/
class Solver{
    private:
        std::string first_string;                                   // first input string
        std::string second_string;                                  // second input string
        std::vector<int>  prefixFunction(std::string  string);  // prefix function for the Knuth-Morris-Pratt method

    public:
        Solver();                                                   // class constructor
        std::string KMPAlgorithm();                                 // Knuth-Morris-Pratt algorithm
        int cycleShiftAlgorithm();                                // the method that determines the cyclic shift
};

/**
 * This method initializes the initial parameters needed to solve the problem (reads the input data)
**/
Solver::Solver(){
    std::cin >> this->first_string;
    std::cin >> this->second_string;
};

/**
 * This method compiles a list of prefix lengths matching suffixes from the passed string
 * Args:
 * - string (std::string) - he string to build the list for
 * Returns:
 * - prefix_indexes (std::vector<int>) -  a list of prefix lengths matching suffixes from the passed string
**/
std::vector<int> Solver::prefixFunction(std::string string){
    std::vector<int> prefix_indexes(string.size(), 0);

    for (size_t i = 1; i < string.size(); i++){
        int j = prefix_indexes[i - 1];

        while ((j > 0) && (string[i] != string[j])){
            j = prefix_indexes[j - 1];
```

```
            }

            if (string[i] == string[j]){
                    j++;
            }

            prefix_indexes[i] = j;
        }

        return prefix_indexes;
    };

    /**
     * This method solves the problem of finding a substring in a
string by the Knuth-Morris-Pratt algorithm
     * Returns:
     * - occurrence_indexes (std::string) - indexes of the occurrence
of the first row in the second
    **/
    std::string Solver::KMPAlgorithm(){
        std::string occurrence_indexes;
        std::string combined_strings = this->first_string + "@" + this-
>second_string;
        std::vector<int>         prefix_indexes         =         this-
>prefixFunction(combined_strings);
        size_t pattern_len = this->first_string.size();

        for  (size_t  index  =  1;  index  <  this->second_string.size();
index++){
            if   (prefix_indexes[pattern_len   +   1   +   index]   ==
pattern_len){
                occurrence_indexes     +=     std::to_string(index     -
pattern_len + 1) + ",";
            }
        }

        if (occurrence_indexes.size() == 0){
            occurrence_indexes = "-1";
        } else {

occurrence_indexes.erase(occurrence_indexes.find_last_not_of(",") + 1);
        }

        return occurrence_indexes;
    };

    /**
     * This method determines whether the first row is a cyclic shift
of the second
     * Returns:
     * - answer (int) - index of the beginning of the second row in the
first
    **/
    int Solver::cycleShiftAlgorithm(){
        int answer = -1;

        if (this->first_string.size() != this->second_string.size()){
            return answer;
```

```
        }

        if (this->first_string.size() == 0 || this->second_string.size()
== 0){
            return answer;
        }

        std::vector<int>  prefix_indexes  =  this->prefixFunction(this-
>second_string);
        size_t first_string_len = first_string.size();
        size_t second_string_len = second_string.size();

        for (size_t index = 0; index < first_string_len; index++){
            size_t first_string_index = index;
            size_t second_string_index = 0;

            while (first_string_index < first_string_len * 2 + index){
                size_t  first_string_real_index  =  first_string_index  %
first_string_len;

                if    (this->first_string[first_string_real_index]    ==
this->second_string[second_string_index]){
                    first_string_index++;
                    second_string_index++;

                    if (second_string_index == second_string_len){
                        answer       =       first_string_index       -
second_string_index;
                        return answer;
                    }
                } else {

                    if (second_string_index > 0){
                        second_string_index                             =
prefix_indexes.at(second_string_index - 1);
                    } else {
                        first_string_index++;
                    }

                }
            }
        }

        return answer;
    };


    /**
     * The main function that implements the solution of the problem
    **/
    int main(){
        Solver* solver = new Solver();
        //std::cout << solver->KMPAlgorithm() << std::endl;
        std::cout << solver->cycleShiftAlgorithm() << std::endl;
        delete solver;
        return 0;
    }
```