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. 1384 _____ Тапеха В.А.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург
2023

Цель работы.

Изучить принцип работы алгоритма Кнута-Морриса-Пратта. Реализовать две программы, использующие этот алгоритм.

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

Выполнение работы.

Задание 1:

Класс *KMP* реализует алгоритм Кнута-Морриса-Пратта (KMP) для поиска всех вхождений шаблона в заданный текст.

Он имеет три поля: *prefix* – значения префикс функции для шаблона, *pattern_matches* – индексы найденных вхождений шаблона в строке, *pattern* – шаблон, который необходимо найти в заданном тексте.

Метод *KMP::findPrefix* вычисляет вектор префиксов для шаблонной строки pattern.

Метод *KMP::kmp* запускает алгоритм КМП на заданном тексте *text* для поиска вхождений шаблонной строки. В этом методе используется вычисленный вектор префиксов, чтобы быстро переходить к следующей позиции, если текущая не соответствует шаблону.

Метод *KMP::printAnswer* печатает ответ на задачу. Если не было найдено шаблона в тексте, то печатается -1.

Задание 2:

Класс *KMP* и представляет собой реализацию алгоритма Кнута-Морриса-Пратта (КМП) для решения задачи проверки, является ли одна строка циклическим сдвигом другой строки.

Он имеет три поля: *shift_string* – сдвинутая строка, *original_string* – оригинальная строка, *prefix_shift_string* – значения префикс-функции для *shift_string*.

Метод *KMP::findPrefix* вычисляет вектор префиксов для строки *shift_string*.

KMP::cyclicKMP - это метод класса *KMP*, который использует алгоритм Кнута-Морриса-Пратта для проверки является ли *shift_string* циклическим сдвигом *original_string*. Метод *KMP::cyclicKMP* возвращает целое число - индекс начала строки *shift_string* в строке *original_string*. Если строка

shift_string не является циклическим сдвигом строки *original_string*, метод возвращает -1.

Метод *KMP::solveTask* вызывает все необходимые для решения задачи методы, проверяет размеры исходных строк и дублирует одну из строк.

Разработанный программный код см. в приложении А.

Вывод.

В рамках лабораторной работы были изучены алгоритм Кнута-Морриса-Пратта и префикс-функция. Для решения задачи были разработаны две программы, основанные на алгоритме Кнута-Морриса-Пратта.

В обеих программах была реализована префикс-функция, которая используется в алгоритме Кнута-Морриса-Пратта для поиска всех вхождений подстроки в строку за линейное время. Этот массив содержит длины максимальных собственных префиксов (подстроки, являющейся начальным фрагментом строки, не совпадающей со всей строкой) каждого префикса строки.

Первая программа использовала алгоритм без каких-либо изменений и находила позицию первого вхождения подстроки в строку. А для решения второй задачи алгоритм был немного изменен таким образом, чтобы больше подходить под требования задачи.

Вторая программа была изменена таким образом, чтобы решать более сложную задачу. В частности, нужно было выяснить является ли подстрока циклическим сдвигом другой строки и была ли она сдвинута на определённое число символов относительно исходной строки. Для этого вторая программа сначала дополняла исходную строку с её конца до двукратной длины, а затем применяла алгоритм Кнута-Морриса-Пратта для поиска подстроки в циклически сдвинутой строке.

Таким образом, в результате выполнения лабораторной работы были получены программы, позволяющие эффективно находить вхождение подстрок в строки с помощью алгоритма Кнута-Морриса-Пратта и префикс-функции. Кроме того, была решена более сложная задача с использованием изменённой версии алгоритма. Обе программы успешно прошли все тестовые задания на платформе stepik.org.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: task1.cpp

```
#include <iostream>
#include <utility>
#include <vector>
#include <experimental/iterator>


// Класс, реализующий алгоритм КМР
class KMP {
public:
    // Конструктор
    explicit KMP(std::string pattern)
    : pattern(std::move(pattern)), prefix(pattern.length()) {}

    // Метод, который ищет значения префикс-функции
    void findPrefix();
    // Метод, реализующий алгоритм КМП
    void kmp(const std::string& text);

    // Возвращает лучшее решение
        [[nodiscard]] std::vector<uint32_t> getSolution() const
{ return pattern_matches; }

    // Печатает ответ на задачу
    void printAnswer() const;
private:
    // Значения префикс-функции
    std::vector<uint32_t> prefix;
    // Индексы найденных вхождения шаблона в тексте
    std::vector<uint32_t> pattern_matches;
    // Шаблон, который необходимо найти в заданном тексте
    const std::string pattern;
};

// Метод KMP::findPrefix вычисляет вектор префиксов для шаблонной
строки pattern
void KMP::findPrefix() {
    prefix.at(0) = 0;

    for (size_t i = 1; i < pattern.length(); ++i) {
        uint32_t k = prefix.at(i - 1);
        while (k > 0 && pattern.at(i) != pattern.at(k)) {
            k = prefix.at(k - 1);
        }

        k += (pattern.at(i) == pattern.at(k));
        prefix.at(i) = k;
    }
}
```

```
    /**
     * Метод KMP::kmp запускает алгоритм КМП на заданном тексте text для поиска
     * вхождений шаблонной строки. В этом методе используется вычисленный вектор
     * префиксов, чтобы быстро переходить к следующей позиции, если текущая не
     * соответствует шаблону.
     **/
    void KMP::kmp(const std::string& text) {
        uint32_t pattern_index = 0;

        for (size_t i = 0; i < text.length(); ++i) {
            while (pattern_index > 0 && pattern.at(pattern_index) != text.at(i)) {
                pattern_index = prefix.at(pattern_index - 1);
            }

            pattern_index += (pattern.at(pattern_index) == text.at(i));

            if (pattern_index == pattern.length()) {
                pattern_matches.push_back(i - pattern.length() + 1);
                pattern_index = prefix.at(pattern_index - 1);
            }
        }
    }

    /**
     * Метод KMP::printAnswer печатает ответ на задачу. Если не было найдено
     * шаблона в тексте, то печатается -1 */
    void KMP::printAnswer() const {
        if (pattern_matches.empty()) {
            std::cout << -1;
            return;
        }

        std::copy(pattern_matches.begin(), pattern_matches.end(),
                  std::experimental::make_ostream_joiner(std::cout, ","));
    }


    int main() {
        std::string text, pattern;

        std::cin >> pattern >> text;

        KMP solve(pattern);
        solve.findPrefix();
        solve.kmp(text);

        solve.printAnswer();
```

```
        return 0;
    }
```

Название файла: task2.cpp

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


    // Класс, решающий задачу
    class KMP {
    public:
        // Конструктор
            explicit KMP(const std::string& original_string, const
std::string& shift_string)
                                    : original_string(original_string),
shift_string(shift_string),
                prefix_shift_string(shift_string.length()) {}

        // Вызывает нужные методы и печатает ответ на задачу
        void solveTask();
    private:
        // Метод, проверяющий размеры строк
              [[nodiscard]] bool checkSizes() const { return
original_string.length() == shift_string.length(); }
        // Метод, который ищет значения префикс-функции для
        void findPrefix();
        // Метод, реализующий циклический алгоритм КМП
        int cyclicKMP();

    private:
        // Сдвинутая строка
        std::string shift_string;
        // Оригинальная строка
        std::string original_string;
        // Значения префикс-функции для строки shift_string
        std::vector<uint32_t> prefix_shift_string;
    };

    // Метод KMP::findPrefix вычисляет вектор префиксов для строки
shift_string.
    void KMP::findPrefix() {
        prefix_shift_string.at(0) = 0;

        for (size_t i = 1; i < shift_string.length(); ++i) {
            size_t k = prefix_shift_string.at(i - 1);
            for (;shift_string.at(i) != shift_string.at(k) && k > 0;)
{
                k = prefix_shift_string.at(k - 1);
            }

            k += (shift_string.at(i) == shift_string.at(k));
            prefix_shift_string.at(i) = k;
        }
```

```
    }

    /**
     * KMP::cyclicKMP() - это метод класса KMP, который использует
алгоритм Кнута-Морриса-Пратта
     * для проверки является ли shift_string циклическим сдвигом
original_string.
     * Метод KMP::cyclicKMP() возвращает целое число - индекс начала
строки shift_string
     * в строке original_string. Если строка shift_string не является
циклическим сдвигом
     * строки original_string, метод возвращает -1.
     **/
    int KMP::cyclicKMP() {
        size_t shift_index = 0;

        for (size_t i = 0; i < original_string.length(); ++i) {
                                for   (;shift_string.at(shift_index)   !=
original_string.at(i) && shift_index > 0;) {
                    shift_index = prefix_shift_string.at(shift_index -
1);
            }

                     shift_index  +=  (shift_string.at(shift_index)  ==
original_string.at(i));

            if (shift_index == shift_string.length()) {
                  return static_cast<int>(i - shift_string.length() +
1);
            }
        }

        return -1;
    }


    /**
     * Метод KMP::solveTask вызывает все необходимые для решения за-
дачи методы, проверяет
     * размеры исходных строк и дублирует одну из строк.
     **/
    void KMP::solveTask() {
        if (!checkSizes()) {
            std::cout << -1;
            return;
        }

        original_string += original_string;
        findPrefix();
        std::cout << cyclicKMP();
    }


    int main() {
        std::string original_string, shift_string;
```

```
    std::cin >> original_string >> shift_string;

    KMP solution(original_string, shift_string);
    solution.solveTask();

    return 0;
}
```