**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МОЭВМ**

# ОТЧЕТ

**по лабораторной работе №4**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: Алгоритм Кнута — Морриса — Пратта**

Студент гр. 8304 _____ Мешков М.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

**Цель работы.**

Изучить принцип работы алгоритма Кнута-Морриса-Пратта на примере решения задачи.

**Постановка задачи.**

Вариант 2. Оптимизация по памяти: программа должна требовать O(m) памяти, где m - длина образца. Это возможно, если не учитвать память, в которой хранится строка поиска.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P ( | P | ≤15000) и текста T ( | T | ≤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

**Описание алгоритма.**

Алгоритм Кнута-Морриса-Пратта позволяет найти все вхождения заданной подстроки в другой строке.

В алгоритме используются префикс-функции. Префикс-функция в заданной позиции строки это длина наибольшего суффикса (но не совпадающего со всей строкой), который является префиксом этой же подстроки. Зная префикс функции в позиции i-1, можно вычислить ее значение в позиции i: пусть k — значение префикс функции для подстроки длины i-1,

если символ в позиции i совпадает с символом в позиции k+1, то префикс в данной позиции будет на 1 больше чем в предыдущей, иначе необходимо рассмотреть префикс подстроки до позиции k.

Алгоритм Кнута-Морриса-Пратта заключается в построении строки вида <слово><отличный символ><текст> и применении к этой строке префикс-функции. Если в какой либо позиции значение префикс функции совпало с длиной искомого слова, значит найдено вхождение в этой позиции. Для построения алгоритма более эффективного по памяти используется модификация: после вычисления префикс функции искомого слова, нет необходимости хранить все последующие значения префикс функции — можно оставлять только последнее для вычисления следующего и для проверки не было ли найдено вхождение.

**Оценка сложности алгоритма.**

Обозначим искомое слово как P, а строку, в которой осуществляется поиск как S.

Для вычисления префикс-функции P необходимо совершить |P| итераций. Внутри каждой итерации находится цикл, количество срабатываний которого в целом не превосходит |P|. Получается оценка O(|P|). Однако необходимо также вычислить значение префикс-функции для S. Итоговая оценка по времени — O(|P| + |S|).

По памяти оценка — O(|P|), т. к. в процессе работы от размера входных данных зависит только размер памяти выделенный под значения префикс-функции.

**Описание функций и структур данных.**

Для решения задачи были реализованы:

Функция countPrefix для вычисления префикс-функции переданной строки, принимает строку, возвращает массив ее значений-префикс функции в каждой позиции строки.

Функция kmpAlgorithm для выполнения  Алгоритма Кнута — Морриса — Пратта, она принимает два параметра-строки — word и text, и ищет word в text, возвращает массив вхождений.

Функция main принимает ввод и выводит ответ. Для подробного вывода процесса работы алгоритма при запуске нужно передать опцию -v, для вывода в файл -fo, для ввода из файла -fi.

**Тестирование.**

Таблица 1 - Тестирование

| Ввод | Вывод |
|---|---|
| ab<br>abab | 0,2 |
| qwer<br>qqwerrrqweqwer | 1,10 |
| qwqw<br>qqwqwqw | 1,3 |
| qwer<br>qwer | 0 |
| t<br>dkf | -1 |

**Выводы.**

В ходе работы был реализован алгоритм Кнута-Морриса-Пратта для нахождения вхождений образа в строке, программа была протестирована на различных входных данных, были получены ожидаемые результаты, была оценена сложность алгоритма - было выяснено, что время работы алгоритма ограничено O(|P| + |S|).

## ПРИЛОЖЕНИЕ А.
## ИСХОДНЫЙ КОД

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

std::ostream *out;
std::istream *in;

bool verbose = false;

// Вычисляет префикс функцию строки s
std::vector<int> countPrefix(const std::string &s) {
    std::vector<int> prefix(s.size(), 0);

    for (int i = 1; i < s.size(); i++) {
        int k = prefix[i - 1];
        for ( ; k > 0 && s[k] != s[i]; k = prefix[k - 1])
            ;
        if (s[k] == s[i])
            k++;
        prefix[i] = k;
    }
    return prefix;
}

// Выполняет Алгоритм Кнута — Морриса — Пратта, возвращает массив
вхождений word в text
std::vector<int> kmpAlgorithm(const std::string &word, const std::string
&text) {
    if (verbose) *out << "Starting KMP algorithm." << std::endl;

    // Массив для сохранения результата функции
    std::vector<int> result;

    if (verbose) *out << "Counting prefix function for word." <<
std::endl;
    auto wordPrefix = countPrefix(word);
    if (verbose) {
        *out << "Word prefix: ";
        for (int i = 0; i < wordPrefix.size(); i++)
            *out << "[" << i << "]" << wordPrefix[i] << " ";
        *out << std::endl;
    };
```

```
    if (verbose) *out << "Start counting prefix function for every
character of the text string." << std::endl;
    if (verbose) *out << "Target prefix value is " << word.size() << "
(word length)." << std::endl;
    int prevK = 0;
    for (int i = 0; i < text.size(); i++) {
        int k = prevK;
        for ( ; k > 0 && word[k] != text[i]; k = wordPrefix[k - 1])
            ;
        if (word[k] == text[i])
            k++;
        prevK = k;

        if (k == word.size())
            result.push_back(i - word.size() + 1);

        if (verbose) *out << "[" << i << "] " << k << (k ==
word.size() ? " - match!" : "") << std::endl;
    }

    return result;
}

int main(int argc , char *argv[]) {
    // Настройка ввода и вывода
    std::ifstream inFile;
    std::ofstream outFile;
    in = &std::cin;
    out = &std::cout;

    // Проверка параметров командной строки (нужно ли включить пошаговый
вывод)
    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-v") == 0)
            verbose = true;
        if (strcmp(argv[i], "-fi") == 0) {
            in = &inFile;
            inFile.open("in.txt");
        }
        if (strcmp(argv[i], "-fo") == 0) {
            out = &outFile;
            outFile.open("out.txt");
        }
    }

    // word будет искаться в text
    std::string word, text;
```

```
    // Ввод
    getline(*in, word);
    getline(*in, text);

    // Обработка
    auto result = kmpAlgorithm(word, text);

    // Вывод результатов
    if (verbose) *out << "Result: ";
    if (!result.empty()) {
        for (int i = 0; i < (int)result.size() - 1; i++) {
            *out << result[i] << ",";
        }
        *out << result.back();
        *out << std::endl;
    }
    else {
        *out << -1 << std::endl;
    }

    return 0;
}
```