МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

ЭЛЕКТРОТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА «ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И АВТОМАТИЗИРОВАННЫХ СИСТЕМ»

**ОТЧЁТ**

**«ЛАБОРАТОРНАЯ №13:**

**АЛГОРИТМ БОЙЕРА — МУРА»**

Дисциплина: «Программирование»

Выполнил:

Студент группы ИВТ-21-2б

Безух Владимир Сергеевич

Проверил:

Доцент кафедры ИТАС

Полякова Ольга Андреевна

Пермь, 2022

Содержание

[1. Постановка задачи 3](#_Toc98953656)

[2. Анализ задачи 4](#_Toc98953657)

[3. Исходный код 6](#_Toc98953658)

[4. Анализ результатов 9](#_Toc98953659)

# Постановка задачи

Найти образец строки в тексте с помощью алгоритма Бойера — Мура.

# 

# Анализ задачи

Алгоритм Бойера — Мура основан на трёх идеях.

1. **Сканирование слева направо, сравнение справа налево.** Совмещаются начала текста и образца строки, проверка начинается с последнего символа образца строки. Если символы совпадают, производится сравнение предпоследнего символа образца строки и т. д. Если все символы образца строки совпали с наложенными символами текста, значит, вхождение образца строки в текст найдено, и выполняется поиск следующего вхождения образца строки.

Если же какой-то символ образца строки не совпадает с соответствующим символом текста, образец строки сдвигается на несколько символов вправо, и проверка снова начинается с последнего символа.

2. **Эвристика стоп-символа.** Предположим, что мы производим поиск слова «колокол». Первая же буква не совпала — «к» (назовём эту букву стоп-символом). Тогда можно сдвинуть образец строки вправо до последней его буквы «к».

Текст: \* \* \* \* \* \* **к** \* \* \* \* \* \*

Образец строки: к о л о к о л

Следующий шаг: к о л о к о л

Если стоп-символа в образце строки нет, образец строки смещается за этот стоп-символ.

Текст: \* \* \* \* \* **а** л \* \* \* \* \* \*

Образец строки: к о л о к о л

Следующий шаг: к о л о к о л

Если стоп-символ «к» оказался за другой буквой «к», эвристика стоп-символа не работает.

Текст : \* \* \* \* **к** к о л \* \* \* \* \*

Образец строки: к о л о к о л

Следующий шаг: к о л о к о л

В таких ситуациях применяется третья идея алгоритма Бойера — Мура.

3. **Эвристика совпавшего суффикса.** Если при чтении образца строки справа налево совпал суффикс S, а символ b, стоящий перед S в образце строки (то есть образец строки имеет вид PbS), не совпал, то эвристика совпавшего суффикса сдвигает образец строки на наименьшее число позиций вправо так, чтобы текст S совпал с образцом строки, а символ, предшествующий в образце строки данному совпадению S, отличался бы от b (если такой символ вообще есть). Например:

Текст: \* \* \* \* \* \* р к а \* \* \* \* \*

Образец строки: с к а л к а л к а

Следующий шаг: с к а л к а л к а

В данном случае совпал суффикс «ка», и образец строки сдвигается вправо до ближайшего «ка», перед которым нет буквы «л».

Текст: \* \* т о к о л \* \* \* \* \*

Образец строки: к о л о к о л

Следующий шаг: к о л о к о л

В данном случае совпал суффикс «окол», и образец строки сдвигается вправо до ближайшего «окол», перед которым нет буквы «л». Если подстроки «окол» в образце строки больше нет, но он начинается на «кол», сдвигается до «кол», и т. д.

# Исходный код

#include <vector>

#include <string>

#include <iostream>

#include <unordered\_map>

// good suffix heuristic

std::vector<int> prefix\_func(const std::string& text)

{

std::vector<int> prefix(text.length());

int k = 0; prefix[0] = 0;

for (int i = 1; i != text.length(); ++i)

{

while (k > 0 && text[k] != text[i])

k = prefix[k - 1];

if (text[k] == text[i])

++k;

prefix[i] = k;

}

return prefix;

}

std::vector<int> find(std::string& text, std::string& pattern)

{

if (text.length() < pattern.length())

return std::vector<int>(1, -1);

if (!pattern.length())

return std::vector<int>(1, text.length());

typedef std::unordered\_map<char, int> TStopTable;

typedef std::unordered\_map<int, int> TSufficsTable;

TStopTable stop\_table;

TSufficsTable suffics\_table;

// bad character heuristic

for (int i = 0; i != static\_cast<int>(pattern.length()); ++i)

stop\_table[pattern[i]] = i;

std::string reverse\_pattern(pattern.rbegin(), pattern.rend());

std::vector<int> prefix = prefix\_func(pattern), reverse\_prefix = prefix\_func(reverse\_pattern);

for (int i = 0; i != pattern.length() + 1; ++i)

suffics\_table[i] = pattern.length() - prefix.back();

for (int i = 1; i != pattern.length(); ++i) {

int j = reverse\_prefix[i];

suffics\_table[j] = std::min(suffics\_table[j], i - reverse\_prefix[i] + 1);

}

std::vector<int> shifts;

for (int shift = 0; shift <= static\_cast<int>(text.length()) - static\_cast<int>(pattern.length());)

{

int position = pattern.length() - 1;

while (pattern[position] == text[position + shift]) {

if (position == 0) { shifts.emplace\_back(shift); break; }

--position;

}

if (position == pattern.length() - 1) {

TStopTable::const\_iterator stop\_symbol = stop\_table.find(text[position + shift]);

int stop\_symbol\_additional = position - (stop\_symbol != stop\_table.end() ? stop\_symbol->second : -1);

shift += stop\_symbol\_additional;

} else shift += suffics\_table[pattern.length() - 1 - position];

}

if (!shifts.empty())

return shifts;

return std::vector<int>(1, -1);

}

int main()

{

std::string text = "TENETENET ABCTENCD TEN ET AENET TENTEN";

std::string first\_pattern = "TEN", second\_pattern = "TENET";

std::cout << "Text: " << text << std::endl;

std::cout << "First pattern: " << first\_pattern << std::endl;

std::vector<int> shifts = find(text, first\_pattern);

for (const auto& shift : shifts)

std::cout << "Shift: " << shift << std::endl;

std::cout << "Second pattern: " << second\_pattern << std::endl;

shifts = find(text, second\_pattern);

for (const auto& shift : shifts)

std::cout << "Shift: " << shift << std::endl;

}

# Анализ результатов

Результаты работы программы (рис. 1).

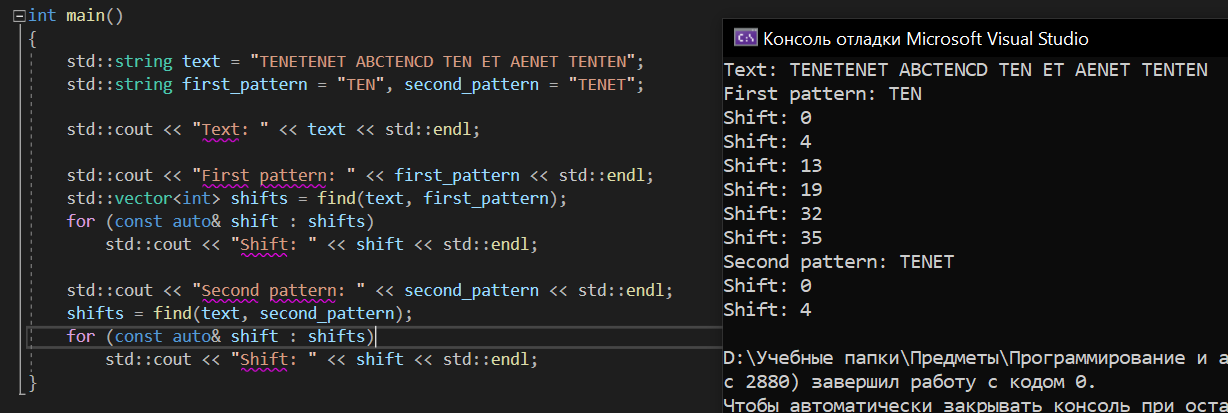


Рисунок — Результаты