**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. 8304 |  | Мухин А. М. |
| Преподаватель |  | Размочаева Н. В. |

Санкт-Петербург

2020

**Цель работы.**

Реализовать алгоритм Кнута-Морриса-Пратта, найти индексы вхождения подстроки в строку.

**Вариант 2.**

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

**Задание.**

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P и текста T найдите все вхождения P в T.

Ввод:

Первая строка – P

Вторая строка – T

Вывод:

Индексы начал вхождений P в T, разделенные пробелом, если P не входит в T, то вывести -1.

**Пример входных данных**

aba

ababa

**Пример выходных данных**

0 2

**Описание алгоритма.**

На вход алгоритму передается строка-образец, вхождения которой нужно найти, и строка-текст, в которой нужно найти вхождения.

Оптимизация – строка-текст считывается посимвольно, в памяти хранится текущий символ.

Алгоритм сначала вычисляет префикс-функцию строки-образца. Далее посимвольно считывается строка-текст. j – счётчик указывающий на позицию рассматриваемую в образце. При каждом совпадении j-го символа образца и очередного символа текста счетчик увеличивается на 1. Если j = размер образца, значит вхождение найдено. Если очередной символ текста не совпал с j-ым символом образца, то в случае, если символ по счёту был первым, то просто сдвигаем считываем следующее значение в строке поиска, а в ином случае сдвигаем образец таким образом, чтобы не пропустить промежуточных вложений.

Сложность алгоритма по операциям: O (m + n), m – длина образца, n – длина текста.

Сложность алгоритма по памяти: O (m), m – длина образца.

**Описание функций.**

std::vector<size\_t> pi\_func(std::string)

Функция вычисления префикс-функции строки. Принимает на вход строку, возвращает массив со значениями префикс-функции.

void KMP(std::istream&, std::ostream&) noexcept

Функция, реализующая алгоритм КМП. Принимает поток ввода и поток вывода. Считывает данные из переданного потока ввода и выводит в переданный поток вывода.

**Тестирование.**

Входные данные:

aabaab

aabaabaaaaaabaabaabaabbaaab

Выходные данные:

Match at: 0 position

Move right at 2 positions instead of 6

Move right at 1 positions instead of 6

Move right at 1 positions instead of 6

Move right at 1 positions instead of 6

Move right at 1 positions instead of 6

Match at: 10 position

Match at: 13 position

Match at: 16 position

Move right at 0 positions instead of 6

Move right at 1 positions instead of 6

Move right at 0 positions instead of 6

Входные данные:

ab

aababaaabaaba

Выходные данные:

Move right at 0 positions instead of 2

Match at: 1 position

Match at: 3 position

Move right at 0 positions instead of 2

Move right at 0 positions instead of 2

Match at: 7 position

Move right at 0 positions instead of 2

Match at: 10 position

Move right at 0 positions instead of 2

**Выводы.**

В ходе выполнения лабораторной работы был реализован алгоритм КМП и функция вычисления префикса строки.

**ПРИЛОЖЕНИЕ А.  
Исходный код**

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

std::vector<size\_t> pi\_func(std::string form) {

size\_t len\_form = form.size();

std::vector<size\_t> local\_pi(len\_form);

for (size\_t i = 1; i < len\_form; ++i) {

size\_t j = local\_pi[i-1];

while (j > 0 && form[i] != form[j]) {

j = local\_pi[j - 1];

}

if (form[i] == form[j]) {

++j;

}

local\_pi[i] = j;

}

return local\_pi;

}

void KMP(std::istream& input, std::ostream& output) noexcept {

bool is\_result = false;

std::string pattern;

input >> pattern;

char tmp = '1';

std::vector<size\_t> pi = pi\_func(pattern);

size\_t j = 0;

size\_t counter = 0;

input.get();

input.get(tmp);

while(input.peek() != EOF) {

while (pattern[j] == tmp) { // пока символы совпадают продолжаем сравнение

input.get(tmp);

++j;

++counter;

}

if (j == pattern.length()) { // по окончанию сравнения проверяем, равен ли счётчик, указывающий

output << "Match at: " << counter - j << " position" << std::endl; // на шаблон длине шаблона (т. е. найдено совпадение). сдвигаем j

j = pi[j-1]; // во избежания пропуска, вложенных совпадений.

is\_result = true;

} else {

if (j != 0) { // если символ не первый, сдвигаем на максимально возможный размер,

j = pi[j-1]; // указанный в массиве pi.

output << "Move right at " << j << " positions instead of " << pattern.length() << std::endl;

} else {

input.get(tmp); // если первый, то считываем новый символ и увеличиваем позицию в строке.

++counter;

}

}

}

if (!is\_result) {

output << -1;

}

output << std::endl;

}

int main() {

int input\_mode = 0;

int output\_mode = 0;

std::cout << "Tap 1 for console input, 0 for file input:";

std::cin >> input\_mode;

std::cout << "Tap 1 for console output, 0 for file output:";

std::cin >> output\_mode;

if (input\_mode == 1 && output\_mode == 1) {

KMP(std::cin, std::cout);

} else if (input\_mode == 1 && output\_mode == 0) {

std::ofstream output("./output.txt", std::ios::out | std::ios::trunc);

KMP(std::cin, output);

} else if (input\_mode == 0 && output\_mode == 1) {

std::ifstream input("./input.txt");

KMP(input, std::cout);

} else if (input\_mode == 0 && output\_mode == 0) {

std::ifstream input("./input.txt");

std::ofstream output("./output.txt", std::ios::out | std::ios::trunc);

KMP(input, output);

} else {

std::cout << "You can tap 0 or 1" << std::endl;

}

return 0;

}