**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Алгоритм Кнута-Морриса-Пратта»**

|  |  |
| --- | --- |
| Студентка гр. 7381 | Машина Ю. Д. |
| Преподаватель | Фирсов М. А. |

Санкт-Петербург

2019

**Цель работы.**

Изучить эффективный алгоритм Кнута-Морриса-Пратта, осуществляющий поиск подстроки в строке: его реализацию, асимптотику по времени работы, асимптотику по требуемой памяти. С помощью данного алгоритма решить ряд задач с учётом варианта индивидуализации.

**Задание 1.**

Реализуйте алгоритм Кнута-Морриса-Пратта и с его помощью для заданных шаблона () и текста () найдите все вхождения в .

**Входные данные задания 1.**

В первой строке входных данных содержится строка шаблона .

Во второй строке входных данных содержится строка текста .

**Пример входных данных задания 1.**

ab

abab

**Выходные данные задания 1.**

Вывод содержит перечисление индексов вхождений шаблона в текст , разделённых запятой. Если не входит в , вывести .

**Пример выходных данных задания 1.**

0,2

**Задание 2.**

Заданы две строки () и ().

Определить, является ли циклическим сдвигом (это значит, что и имеют одинаковую длину и состоит из суффикса , склеенного с префиксом ).

**Входные данные задания 2.**

В первой строке входных данных содержится строка .

Во второй строке входных данных содержится строка .

**Пример входных данных задания 2.**

defabc

abcdef

**Выходные данные задания 2.**

Единственная строка выходных данных содержит индекс начала строки в , если является циклическим сдвигом , иначе . Если возможно несколько сдвигов, вывести первый индекс.

**Пример выходных данных задания 2.**

3

**Дополнительные сведения.**

Во всех заданиях предполагается, что символы строк принадлежат латинскому алфавиту.

**Индивидуализация.**

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

**Теоретические сведения.**

Префикс-функция (англ. prefix-function) от строки – массив длин наибольших бордеров для каждой позиции этой строки.

Префикс — это подстрока, начинающаяся с начала строки.

Суффикс — это подстрока, заканчивающаяся в конце строки.

Бордер – префикс подстроки, совпадающий с её суффиксом.

Префикс функция от строки в позиции определяется следующим образом:

Если такое не найдено, то .

Алгоритм Кнута-Морриса-Пратта заключается в том, что необходимо склеить строку шаблона и строку текста , разделив их специальным символом , не содержащимся ни в шаблоне, ни в тексте. Найдём значения префикс функции для всех символов склеенной строки . Линейно проверим все значения префикс функции: если какое-то из значений совпадает с длиной шаблона , то позиция является позицией вхождения шаблона в строку , что соответствует позиции в исходном тексте .

Для выполнения второго задания лабораторной работы достаточно сделать 2 вещи:

1. Если длины строк и не совпадают, то, очевидно, шаблон не является циклической перестановкой текста.
2. Удвоим длину текста: . Применим алгоритм Кнута-Морриса-Пратта для шаблона и текста .

**Описания алгоритмов.**

Алгоритм вычисления префикс функции:

1. Положим ;
2. Если , то и повторяем шаг 2 для следующего , если не достигнут конец строки.
3. Подберём такое , что для этого примем . Если пробуем , пока . Возвращаемся к шагу 2.

Алгоритм Кнута-Морриса-Пратта:

1. Сформируем строку .
2. Посчитаем префикс функцию для строки .
3. Просмотрим все значения префикс функции: если значение совпадает с длиной шаблона , то сохраним индекс – индекс вхождения заданной подстроки.

Алгоритм Кнута-Морриса-Пратта (с экономией по памяти):

1. Найдём префикс функцию от шаблона .
2. Будем проходить по тексту (индекс ) и шаблону (индекс ) независимо.
3. Если , то увеличиваем , иначе, если , делаем переход .
4. Увеличиваем , пока не достигнем конца строки.

**Асимптотика алгоритма.**

Префикс-функцию можно посчитать за линейное время , где – длина исходной строки. На каждом шаге увеличивается не больше, чем на единицу, при этом максимальное значение . Поскольку в шаге 3 величина убывает, получается, что не может уменьшиться больше, чем раз, что, в конечном счёте, и определяется итоговую асимптотику.

Чтобы выполнить алгоритм Кнута-Морриса-Пратта необходимо построить префикс функцию для строки , что можно сделать за линейное время от размеров исходных и , то есть за , а затем проверить все найденные значения с позиции до , то есть ровно значений, или в -нотации: . Данные оценки дают итоговую оценку .

Затраты по памяти составляются из необходимости хранить строку шаблона, строку текста и значения префикс функции для строки , то есть асимптотика аналогичная, что и для затрат по времени: . Это классическая оценка по памяти для Кнута-Морриса-Пратта. Представленный алгоритм требует память для шаблона , текста , префикс функции от . Если не учитывать память, необходимую для хранения строки , то затраты по памяти, очевидно, .

**Описание функций и структур данных.**

std::vector<size\_t> prefix\_function(const std::string& text);

Данная функция принимает на вход строку по константной ссылке и возвращает вектор целых чисел длины входной строки, содержащий значения префикс функции для каждого индекса строки.

std::vector<size\_t> knuth\_morris\_pratt(const std::string& text, const std::string& pattern);

Данная функция принимает на вход по константной ссылке строку текста и строку шаблона . Возвращаемое значение данной функции – вектор индексов всех вхождений шаблона в текст. Если вхождений не найдено, то вектор пуст.

class KnuthMorrisPrattOptimized

Класс предназначен для использования алгоритма Кнута-Морриса-Пратта и содержит единственный метод:

static std::vector<size\_t> find(const std::string& text, const std::string& pattern);

Данный метод принимает на вход по константной ссылке строку текста и строку шаблона . Возвращаемое значение данного метода – вектор индексов всех вхождений шаблона в текст. Если вхождений не найдено, то вектор пуст.

**Тестирование.**

Таблица 1 – Результаты тестирования для задания 1.

|  |  |  |
| --- | --- | --- |
| № | Входные данные | Выходные данные |
| 1 | ab  abab | 0,2 |
| 2 | the  mother\_father\_sister\_brother | 2,9,24 |
| 3 | needle  find\_a\_needle\_in\_a\_haystack | 7 |
| 4 | ad  sfsffbab | -1 |

Таблица 2 – Результаты тестирования для задания 2.

|  |  |  |
| --- | --- | --- |
| № | Входные данные | Выходные данные |
| 1 | defabc  abcdef | 3 |
| 2 | aaaa  aa | -1 |
| 3 | abcdef  fedcba | -1 |
| 4 | mother father sister brother  father\_sister\_brothermother\_ | 7 |

**Выводы.**

Алгоритм Кнута-Морриса-Пратта (КМП-алгоритм) – эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объёма входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно.

**ПРИЛОЖЕНИЕ А**

**АЛГОРИТМА КНУТА-МОРРИСА-ПРАТТА**

#include "pch.h"

#include <iostream>

#include <thread>

#include <string>

#include <vector>

class KnuthMorrisPrattOptimized

{

public:

static std::vector<size\_t> find(const std::string& text, const std::string& pattern);

};

std::vector<size\_t> prefix\_function(const std::string& text) {

std::cout << "--calculation of prefix function--" << std::endl;

std::cout << "prefix function value for the 1st character: '" << text.front() << "' = 0" << std::endl;

std::vector<size\_t> result(text.length());

for (size\_t i = 1; i < text.length(); ++i)

{

std::cout << "current character: '" << text[i] << "' (index i = " << i << ")" << std::endl;

// by default, result is filled with zeros

size\_t j = result[i - 1];

std::cout << "last known prefix-function value j: " << j << std::endl;

// осуществляем возврат по известным значениям префикс функции

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

{

std::cout << "j="<<j<<" is larger than 0, and ";

std::cout << "s[i=" << i << "]" << " = '" << text[i] << "' isn't equal to s[j=" << j << "] = '" << text[j] << "', therefore j is decremented: j = ";

j = result[j - 1];

std::cout << j << std::endl;

}

// увеличиваем значение префикс функции на единицу

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

{

++j;

std::cout <<"text[i="<<i<<"]"<< " = '"<< text[i] <<"' is equal to "<<"text[j="<<j-1<<"] = '"<<text[j-1]<<"', therefore j is incremented: j = " << j << std::endl;

}

std::cout << "prefix-function value found: " << j << std::endl << std::endl;

result[i] = j;

}

std::cout << "--calculation of prefix-function complete--" << std::endl;

return result;

}

std::vector<size\_t> KnuthMorrisPrattOptimized::find(const std::string& text, const std::string& pattern)

{

std::vector<size\_t> entries;

size\_t p = pattern.length();

size\_t t = text.length();

if (p > t || p == 0)

return entries;

std::vector<size\_t> prefix = prefix\_function(pattern);

std::cout << "prefix-function values: " << std::endl;

for (size\_t i = 0; i < pattern.length(); ++i)

std::cout << pattern[i] << " ";

std::cout << std::endl;

for (size\_t i = 0; i < prefix.size(); ++i)

std::cout << prefix[i] << " ";

std::cout << std::endl;

for (size\_t i = 0, q = 0; i < t; ++i)

{

while (true)

{

if (text[i] == pattern[q])

{

++q;

if (q == p)

entries.push\_back(i + 1 - p);

break;

}

if (q == 0) //so that pattern's siterator wouldn't be incremented unless a potential entry is present

break;

q = prefix[q - 1];

}

}

return entries;

}

int main() {

std::string pattern;

std::cin >> pattern;

std::string text;

std::cin >> text;

std::vector<size\_t> entries = KnuthMorrisPrattOptimized::find(text, pattern);

// no entries found

if (entries.empty())

{

std::cout << -1 << std::endl << std::endl;

return 0;

}

std::cout << "\nEntries of '"<<pattern<< "' start on the following indexes of '" << text << "':"<< std::endl;

for (size\_t i = 0; i < entries.size() - 1; ++i)

std::cout << entries[i] << ',';

std::cout << entries.back() << std::endl << std::endl;

return 0;

}