**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Поиск образца в тексте: алгоритм Рабина-Карпа.

|  |  |  |
| --- | --- | --- |
| Студент гр. 0381 |  | Дзаппала Д. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2021

## Цель работы.

Реализовать алгоритм нахождения подстроки в строке, используя алгоритм Рабина-Карпа.

## Задание.

Напишите программу, которая ищет все вхождения строки Pattern в строку Text, используя алгоритм Карпа-Рабина.

На вход программе подается подстрока Pattern и текст Text. Необходимо вывести индексы вхождений строки Pattern в строку Text в возрастающем порядке, используя индексацию с нуля.

**Ограничения**

1 ≤ |Pattern| ≤ |Text| ≤ 5 · 105.

Суммарная длина всех вхождений образца в текста не превосходит 108. Обе строки содержат только буквы латинского алфавита.

## Выполнение работы.

Была реализована хеш-функция, принимащая на вход строку, и возвращающая хеш (= числовое значение строки) по модулю. Разработанный алгоритм Рабина-Карпа работает по принипу, что сначала он считает хеш подстроки (паттерна), после чего начинает поиск в исходной строке таким образом, что он каждый раз берет кусок строки длины со строку паттерна, проверяет какой у него хеш. Если хеши равны (с тем что у паттерна), то уже проверяются символы на то что они равны. Если все равны, в результативный вектор int’ов записывается индекс исходной строки, на которой мы начали проверять подстроку. После чего в конце возвращается результативный вектор.

## Выводы.

Был реализован алгоритм нахождения подстроки в строке.

# Приложение А Исходный код программы

Название файла: src.cpp

#include<iostream>

#include<vector>

#include<cmath>

#include<string>

#include<cassert>

#define TESTING

uint32\_t hash(const std::string& pattern, uint32\_t q){

uint32\_t p\_len = pattern.length();

uint32\_t h = 0;

uint32\_t b = 10;

for (size\_t i = 0; i < p\_len; ++i)

h += (h \* b + int(pattern[i])) % q;

return h;

}

std::vector<uint32\_t> RabinKarpSearch(const std::string& Text, const std::string& Pattern){

uint32\_t t\_len = Text.length();

uint32\_t p\_len = Pattern.length();

std::vector<uint32\_t> result;

uint32\_t q = 997;

uint32\_t p\_hash = hash(Pattern, q);

for (size\_t i = 0; i <= t\_len - p\_len; ++i){

std::string tmp;

for (size\_t j = i; j < i + p\_len; ++j){

tmp.push\_back(Text[j]);

}

if (p\_hash == hash(tmp, q)){

for (size\_t k = 0; k < p\_len && Pattern[k] == tmp[k]; ++k)

if (k == p\_len - 1)

result.push\_back(i);

}

}

return result;

}

void test(const std::vector<std::pair<std::pair<std::string, std::string>, std::vector<uint32\_t>>>& testCases){

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

std::cout << "Test#" << i+1 << ":" << std::endl;

std::cout << "Pattern - " << testCases[i].first.first << ", text - " << testCases[i].first.second << " ==> ";

assert(RabinKarpSearch(testCases[i].first.second, testCases[i].first.first) == testCases[i].second && "Error!");

std::cout << " Correct!" << std::endl;

}

}

int main(){

#ifdef TESTING

test({{{"bra", "abracadabra"}, {1, 8}},

{{"Iron", "Iron man has the best Iron suite."}, {0, 22}},

{{"text", "this text is just text."}, {5, 18}},

{{"something", "in this sentense there arent patterns..."}, {}}

});

return 0;

#endif

std::string text, pattern;

std::getline(std::cin, pattern);

std::getline(std::cin, text);

std::vector<uint32\_t> result = RabinKarpSearch(text, pattern);

for (auto& el: result)

std::cout << el << " ";

std::cout << std::endl;

return 0;

}