МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное

образовательное учреждение

высшего образования

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

**по дисциплине**

**“ Алгоритмы построение и анализ”**

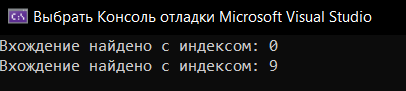
Выполнил: студент гр. ФИб-3302-51-00 ё. \_\_\_\_\_\_\_\_\_\_\_

Проверил: доцент кафедры ПМиИ Разова Е. В. \_\_\_\_\_\_\_\_\_\_\_

Киров 2020

***Задание 1. Алгоритм Рабина-Карпа***

Реализовать алгоритм Рабина-Карпа поиска подстроки в строке.



#include <iostream>

#include <clocale>

using namespace std;

#define d 256 // количество символов во входном алфавите

void search(char pat[], char txt[], int q)

{

setlocale(0, "");

int M = strlen(pat);

int N = strlen(txt);

int i, j;

int p = 0; // хеш-значение для шаблона

int t = 0; // хеш-значение для txt

int h = 1;

// Значение h будет "pow (d, M-1)% q".

for (i = 0; i < M - 1; i++)

h = (h \* d) % q;

// Вычислить хеш-значение шаблона и первого окна текста

for (i = 0; i < M; i++)

{

p = (d \* p + pat[i]) % q;

t = (d \* t + txt[i]) % q;

}

// Поочередно перемещаем подстроку по тексту

for (i = 0; i <= N - M; i++)

{

// проверка хеш-значений текущего окна текста и подстроки. Если значения хеш-функции совпадают, то только

// проверить символы по одному

if (p == t)

{

/\* провека символы по одному\*/

for (j = 0; j < M; j++)

{

if (txt[i + j] != pat[j])

break;

}

// if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]

if (j == M)

cout << "Вхождение найдено с индексом: " << i << endl;

}

// Вычисление значение хеш-функции для следующего окна текста: удалить начальную цифру, добавить конечную цифру

if (i < N - M)

{

t = (d\*(t - txt[i] \* h) + txt[i + M]) % q;

// получение отрицательного значение t, преобразовав его в положительное.

if (t < 0)

t = (t + q);

}

}

}

int main()

{

char txt[] = "ABCD ABC ABCD";

char pat[] = "ABCD";

int q = 110; //хэш значение

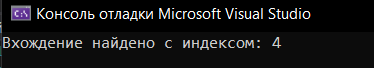
search(pat, txt, q);

return 0;

}

***Задание 2. Алгоритм Бойера-Мура-Хорспула***

Реализовать алгоритм Бойера-Мура-Хорспула поиска подстроки в строке.



#include <iostream>

#include <algorithm>

#include <clocale>

using namespace std;

// Функция предварительной обработки для плохих символов Бойера Мура

void badCharCheck(string str, int size,

int badchar[256])

{

int i;

// инициализация всех вхождения как -1

for (i = 0; i < 256; i++)

badchar[i] = -1;

//заполнение фактических значений последнего появления символа

for (i = 0; i < size; i++)

badchar[(int)str[i]] = i;

}

// Функция поиска шаблонов, использующая исходя из плохих символов алгоритма Бойера Мура.

void search(string txt, string pat)

{

int m = pat.size();

int n = txt.size();

int badchar[256];

//заполнение массива неверных символов, с помощью предварительной обработки

badCharCheck(pat, m, badchar);

int s = 0; //s - сдвиг шаблона относительно текста

while (s <= (n - m))

{

int j = m - 1;

// уменьшение индекса j шаблона, пока символы шаблона и текста не будут соответствовать s

while (j >= 0 && pat[j] == txt[s + j])

j--;

// Если шаблон присутствует в текущем сдвиге, то индекс j станет -1 после

if (j < 0)

{

cout << "Вхождение найдено с индексом: " << s << endl;

//сдвиг паттерна так, чтобы следующий символ в тексте выравнялся с последним появлением его в образце.

s += (s + m < n) ? m - badchar[txt[s + m]] : 1;

}

else

/\*сдвиг для проверки плохого символа что бы совпадал с последним входением в шаблоне

Функция max для проверки положитльного сдвига

Получение отрицательного сдвига возможно, если последнее появление плохого символа в шаблоне

находится справа от текущего символа. \*/

s += max(1, j - badchar[txt[s + j]]);

}

}

int main()

{

setlocale(0, "");

string txt = "ABAAABCD";

string pat = "ABC";

search(txt, pat);

return 0;

}

***Задание 3. Сравнительный алгоритмов поиска подстроки в строке***

Разработать систему тестов и провести сравнительный анализ четырех рассмотренных алгоритмов поиска подстроки в строке (наивный алгоритм, алгоритм Кнута-Морриса-Пратта, алгоритм Рабина-Карпа, алгоритм Бойера-Мура-Хорспула) по времени работы, числу сдвигов подстроки, области применимости алгоритмов.

**Сравнение по времени работы**

|  |  |  |
| --- | --- | --- |
| Кнута-Морриса-Пратта | Рабина-Карпа | Бойера-Мура-Хорспула |
| O(m(n-m+1)) | O(n+m) | O(n/m) |

m-длина подстроки

n-длина строки

Алгоритм Кнута-Морриса-Пратта

Наихудшая сложность алгоритма наивного составляет O (m (n-m + 1)). Временная сложность алгоритма KMP составляет O (n) в худшем случае.

Алгоритм Рабина-Карпа

Среднее время работы алгоритма Рабина-Карпа в лучшем случае составляет O (n + m), а в худшем случае - O (nm). Худший случай алгоритма Рабина-Карпа возникает, когда все символы шаблона и текста совпадают, так как хеш-значения всех подстрок txt [] совпадают с хеш-значением pat []. Например, pat [] = «AAA» и txt [] = «AAAAAAA».

Алгоритм Бойера-Мура-Хорспула

Мы предварительно обрабатываем шаблон и сохраняем последнее вхождение каждого возможного символа в массиве размером, равным размеру алфавита. Если символ отсутствует, это может привести к сдвигу на m (длина шаблона). Следовательно, эвристика плохих символов в лучшем случае занимает O (n / m) времени.  
  
**Сравнение по применимости**  
  
Алгоритм Рабина-Карпа лучше при поиске большого текста, нахождение нескольких совпадений шаблонов, как обнаружение плагиата.

Алгоритм Бойера-Мура-Хорспула лучше, когда подстрока относительно большая, с алфавитом среднего размера и с большим словарным запасом. И это плохо работает с двоичными строками или очень короткими подстроками.

Алгоритм Кнута-Морриса-Пратта хорош для поиска внутри меньшего алфавита, например, в биоинформатике или поиске в двоичных строках. И он работает медленнее, если алфавит увеличивается.