Министерство образования и науки Российской федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

«\_\_\_\_\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2015 г.

Отчет

по лабораторной работе № 2

Поиск подстрок

(название лабораторной работы)

по дисциплине «Алгоритмы и структуры данных»

**ЛР 09.03.04.11.000 О**

Студент группы ПИ-32 Глушков Г.Г.

(И.О. Фамилия)

Преподаватель преподаватель Н.Д. Бубнова

(должность, ученое звание) (И.О. Фамилия)

Барнаул 2015

*Задание. Вариант 13.*

Формулировка задачи:

Сравнить по быстродействию алгоритм Бойера-Мура и алгоритм Рабина-Карпа при поиске в тексте одного из k образцов

* Алгоритм решения:
* Построчное чтение текста с файла.
* Ввод шаблона с клавиатуры
* Поиск в тексте с помощью алгоритма Бойера-Мура:

Алгоритм сравнивает символы шаблона  справа налево, начиная с самого правого, один за другим с символами исходной строки . Если символы совпадают, производится сравнение предпоследнего символа шаблона и так до конца. Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и поиск окончен. В случае несовпадения какого-либо символа (или полного совпадения всего шаблона) он использует правило сдвига плохого символа, чтобы сдвинуть позицию для начала сравнения вправо.

**Правило сдвига плохого символа**

В таблице плохих символов указывается последняя позиция в шаблоне (исключая последнюю букву) каждого из символов алфавита. Для всех символов, не вошедших в шаблон, пишем . Предположим, что у нас не совпал символ  из текста на очередном шаге с символом из шаблона. Очевидно, что в таком случае мы можем сдвинуть шаблон до первого вхождения этого символа  в шаблоне, потому что совпадений других символов точно не может быть. Если в шаблоне такого символа нет, то можно сдвинуть весь шаблон полностью.

* Поиск в тексте с помощью алгоритма Рабина-Карпа:

Алгоритм начинается с подсчета  и , а также с подсчета , для ускорения ответов на запрос.

Для  вычисляется  и сравнивается с . Если они оказались равны, то образец  скорее всего содержится в строке  начиная с позиции , хотя возможны и ложные срабатывания алгоритма. Чтобы такие срабатывания исключить вовсе, применяем явное сравнение строк. Проверка займет время, равное длине образца, но полностью исключит возможность ложного срабатывания.

* Вывод результатов и времени работы.
* Текст программы:

Файл BoyerMoor.java

package stringsearching;

/\*\*

\*

\* **@author** spirifoxy

\*/

public class **BoyerMoore** {

private int[] symbolsShift; //таблица смещений

private String pattern;

//конструктор

public **BoyerMoore**(String pattern) {

final int alphabetSize = 256;

this.pattern = pattern;

symbolsShift = new int[alphabetSize];

for (int i = 0; i < alphabetSize; i++) {

symbolsShift[i] = -1;

}

//формируем смещения

for (int i = 0; i < pattern.length(); i++) {

symbolsShift[pattern.charAt(i)] = i;

}

}

/\*\*

\*

\* **@param** text Строка, в которой осуществляется поиск

\* **@return** result Строка, содержащая все позиции вхождения шаблона

\*/

public String **search**(String text) {

String result = "";

int pLength = pattern.length();

int tLength = text.length();

int skip;

for (int i = 0; i <= tLength - pLength; i += skip) {

skip = 0;

for (int j = pLength - 1; j >= 0; j--) { //проверка с последнего сиивола

if (pattern.charAt(j) != text.charAt(i + j)) {

//смещение или на 1 символ, или на длину проверенной части шаблона

skip = Math.*max*(1, j - symbolsShift[text.charAt(i + j)]);

break;

}

}

if (skip == 0) {

result += i + " "; // найдено

skip = pLength; // смещение на длину шаблона

}

}

return result;

}

}

Файл RabinKarp.java

package stringsearching;

import java.math.BigInteger;

import java.util.Random;

/\*\*

\*

\* **@author** spirifoxy

\*/

public class **RabinKarp** {

private String pattern; //непосредственно шаблон

private int pLength; //и его длина

private long primeNum; //рандомное большое простое число

private long patHash; //хэш шаблона

final int alphSize;

private long alphSizeExp; // alphSize^(pLength-1) % primeNum

public **RabinKarp**(String pattern) { //конструктор

this.pattern = pattern;

pLength = pattern.length();

primeNum = longRandomPrime();

alphSize = 256;

alphSizeExp = 1;

for (int i = 1; i <= pLength - 1; i++) {

alphSizeExp = (alphSize \* alphSizeExp) % primeNum;

}

patHash = hash(pattern, pLength); //вычисление хэша шаблона

}

//получение случайного простого числа

private long **longRandomPrime**() {

BigInteger prime = BigInteger.*probablePrime*(31, new Random());

return prime.longValue();

}

//хэширование

private long **hash**(String key, int pLength) {

long h = 0;

for (int j = 0; j < pLength; j++) {

h = (alphSize \* h + key.charAt(j)) % primeNum;

}

return h;

}

//проверка на посимвольное соответствие

private boolean **check**(String text, int i) {

for (int j = 0; j < pLength; j++) {

if (pattern.charAt(j) != text.charAt(i + j)) {

return false;

}

}

return true;

}

public String **search**(String text) {

String result = "";

int N = text.length();

if (N < pLength) {

return result;

}

long txtHash = hash(text, pLength);

// проверка на совпадение в позиции 0

if ((patHash == txtHash) && check(text, 0)) {

result += "0 ";

}

// проверка на совпадение по хэшу; если есть - посимвольная проверка

for (int i = pLength; i < N; i++) {

// сдвиг на 1 символ по тексту

txtHash = (txtHash + primeNum - alphSizeExp \* text.charAt(i - pLength) % primeNum) % primeNum;

txtHash = (txtHash \* alphSize + text.charAt(i)) % primeNum;

// точная проверка

int offset = i - pLength + 1;

if ((patHash == txtHash) && check(text, offset)) {

result += offset + " ";

}

}

return result;

}

}

Файл StringSearch.java

package stringsearching;

import java.io.\*;

/\*\*

\* **<p>**

\* **13.** Сравнить по быстродействию алгоритм Бойера-Мура и алгоритм Рабина-Карпа

\* при поиске в тексте одного из k образцов**</p>**

\*

\* **@author** spirifoxy

\*/

public class **StringSearching** {

private static long ***getCurrentTime***() { //текущее время в мс

return System.*currentTimeMillis*();

}

private static long ***countPastTime***(long time) { //прошедшее время в мс

return *getCurrentTime*() - time;

}

public static void ***main***(String[] args) throws IOException {

long timeOut;

BoyerMoore bm;

RabinKarp rk;

File textFile;

BufferedReader in;

String text = "";

String pattern, line;

textFile = new File("text.txt");

in = new BufferedReader(new FileReader(textFile));

while ((line = in.readLine()) != null) {

text += line;

}

in = new BufferedReader(new InputStreamReader(System.*in*));

pattern = in.readLine();

timeOut = *getCurrentTime*();

bm = new BoyerMoore(pattern);

String result = bm.search(text);

System.*out*.println(result);

timeOut = *countPastTime*(timeOut);

System.*out*.println("BM time: " + timeOut);

timeOut = *getCurrentTime*();

rk = new RabinKarp(pattern);

result = rk.search(text);

System.*out*.println(result);

timeOut = *countPastTime*(timeOut);

System.*out*.println("RK time: " + timeOut);

in.close();

}

}

* Тесты



