Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет: «Информационных технологий и прикладной математики»

Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа № 4**

**по курсу «Дискретный анализ»**

|  |  |
| --- | --- |
| Cтудент: | Королев И.М. |
| Группа: | М8О-208Б-19 |
| Преподаватель: | Капралов Н.С. |
| Дата: |  |
| Оценка: |  |

Москва 2020

# 1. Постановка задачи

**Вариант алгоритма:** Поиск одного образца, основанный на построении Z-блоков.

**Вариант алфавита:** Слова не более 16 знаков латинского алфавита (регистронезависимые).

Искомый образец задаётся на первой строке входного файла. Затем следует текст, состоящий из слов, в котором нужно найти заданный образец.

Никаких ограничений на длину строк, равно как и на количество слов или чисел в них, не накладывается.

В выходной файл нужно вывести информацию о всех вхождениях искомого образца в обрабатываемый текст. При выводе информации следует вывести два числа через запятую: номер строки и номер слова в строке, с которого начинается найденный образец. Нумерация начинается с единицы. Номер строки в тексте должен отсчитываться от его реального начала (то есть, без учёта строки, занятой образцом).

# 2. Описание программы

Для выполнения лабораторной работы был создан класс TWord, который содержит в себе слово, его позицию в строке и номер строки, на которой оно находится. Был создан вектор, в который помещаются все слова, которые подаются на вход. Программа читает входные данные. Первой строкой вводится образец (pattern). После того, как первая строка была прочитана, в вектор слов помещается разделитель $ для того, чтобы разделять образец и текст. Далее программа считывает весь текст. После того, как весь текст был прочитан, выполняется Z-функция, которая проверяет вхождение шаблона в текст и записывает в результирующий вектор число букв, которое совпадает у текста и образца с указанного места в тексте. После выполнения Z-функции выполняется проход по результирующему вектору. Если число в векторе встретится число, которое совпадает с размером образца, то выводится номер строки и номер слова в этой строке, из вектора, содержащего весь текст с таким же индексом, что и в результирующем векторе. Номер строки и номер слова и будут обозначать с какого места совпадают образец и текст.

Рассмотрим две реализации Z-функции.

1. Тривиальный алгоритм. Простая реализация за , где – длина строки. Для каждой позиции выполняется перебор для неё, пока не обнаружится несовпадение. То есть для каждой позиции образец прикладывается к тексту.
2. Эффективный алгоритм поиска. Z-блок – подстрока с началом в позиции и длиной . Для выполнения алгоритма создаются две переменные и – начало и конец Z-блока строки S максимальной позицией конца (среди всех Z-блоков выбирается наибольший). Изначально переменные и равны нулю. Пусть известны значения Z-функции от до . Нужно найти . Может быть два случая:
3. . В таком случае происходит прохождение по строке и сравниваются слова на позициях и . Пусть – первая позиция в S для которой не выполняется равенство , тогда – Z-функция для позиции . Тогда , . В данном случае будет определено корректное значение , потому что оно определяется путём сравнения с начальными словами строки.
4. . Сравниваются  и . Если right меньше, то необходимо пройти по строке с позиции right и вычислить значение .

Описанный выше эффективный алгоритм работает за , где m – длина образца, а n – длина текста. Это является верным, потому что каждая позиция проходится не больше, чем 2 раза: при попадании в диапазон от до и при высчитывании Z-функции циклом.

# 

# 3. Исходный код

**search.hpp**

#pragma once

#include <iostream>

#include <vector>

#include <cstring>

struct TWord {

    std::string Word;

    unsigned long long StringId;

    unsigned long long WordId;

    TWord(std::string &word, unsigned long long string\_id,

unsigned long long word\_id): Word(word), StringId(string\_id), WordId(word\_id) {}

    TWord(): Word(""), StringId(0), WordId(0) {}

};

void WordToLower(std::string &string);

std::vector<unsigned long long> ZFunction(const std::vector<TWord> & text);

**search.hpp**

#include "search.hpp"

#include <algorithm>

#include <iostream>

void WordToLower(std::string &string) {

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

        string[i] = std::tolower(string[i]);

    }

}

bool operator == (const TWord &left\_value, const TWord &right\_value) {

    if (left\_value.Word == right\_value.Word) {

        return true;

    }

    return false;

}

bool operator != (const TWord &left\_value, const TWord &right\_value) {

    return !(left\_value == right\_value);

}

std::vector<unsigned long long> ZFunction(const std::vector<TWord> &text) {

    unsigned long long size = text.size();

    unsigned long long left = 0;

    unsigned long long right = 0;

    std::vector<unsigned long long> z\_function(size);

    for (unsigned long long i = 1; i < size; ++i) {

        if (i <= right) {

            z\_function[i] = std::min(z\_function[i-left], right - i + 1);

        }

        while ((z\_function[i] + i < size) && (text[z\_function[i]].Word == text[z\_function[i] + i].Word)) {

            ++z\_function[i];

        }

        if (i + z\_function[i] - 1 > right) {

            left = i;

            right = z\_function[i] + i - 1;

        }

    }

    return z\_function;

}

**main.hpp**

#include <iostream>

#include "search.hpp"

#include <string>

#include <sstream>

int main() {

    std::vector<TWord> text;

    unsigned long long pattern\_size = 0;

    unsigned long long text\_size;

    unsigned long long line = 0;

    unsigned long long word\_number = 1;

    std::string input;

    while (std::getline(std::cin, input)) {

        std::string word;

        std::stringstream temp\_string(input);

        while (temp\_string >> word) {

            WordToLower(word);

            TWord current\_word(word, line, word\_number);

            text.push\_back(current\_word);

            if (line != 0) {

                text\_size++;

            } else {

                pattern\_size++;

            }

            word\_number++;

        }

        if (line == 0) {

            std::string delimiter = "$";

            TWord current\_word(delimiter, line, word\_number);

            text.push\_back(current\_word);

        }

        line++;

        word\_number = 1;

    }

    auto start = high\_resolution\_clock::now();

    std::vector<unsigned long long> res = ZFunction(text);

    auto end = high\_resolution\_clock::now();

    auto duration = duration\_cast<microseconds>(end-start);

    std::cout << "Time: " << duration.count() << " microseconds" << "\n";

}

# 

# 4. Консоль

root@Harry:~/work/MAI/DA/da\_lab4# make

g++ --std=c++17 -Wall -c -O3 main.cpp

g++ --std=c++17 -Wall -c -O3 search.cpp

g++ main.o search.o -o solution

root@Harry:~/work/MAI/DA/da\_lab4# cat test\_01.txt

cat dog cat dog bird

CAT dog CaT Dog Cat DOG bird CAT

dog cat dog bird

root@Harry:~/work/MAI/DA/da\_lab4# ./solution < test\_01.txt

1, 3

1, 8

root@Harry:~/work/MAI/DA/da\_lab4# cat test\_02.txt

dog bird bird cat cat bird

python

kiss

umbrella jump python bird umbrella dog bird bird cat cat

bird dog bird bird cat

cat bird hello

root@Harry:~/work/MAI/DA/da\_lab4# ./solution < test\_02.txt

5, 6

6, 2

root@Harry:~/work/MAI/DA/da\_lab4# cat test\_03.txt

dog bird cat cat bird

umbrella cat dog

bird cat cat bird get umbrella cat dog bird cat cat bird bird dog bird cat cat bird python

dog

bird

cat cat

bird dog cat

jump

root@Harry:~/work/MAI/DA/da\_lab4# ./solution < test\_03.txt

2, 3

3, 8

3, 14

4, 1

# 

# 5. Тест производительности

Было выполнено сравнение алгоритма построения Z-блоков с помощью Z-функции с алгоритмом КМП (который считывает текст посимвольно). Были проведены тесты на , и словах, которые находились в тексте. Шаблон состоял из 3 – 20 слов.

root@Harry:~/work/MAI/DA/da\_lab4# make

g++ --std=c++17 -Wall -c -O3 main.cpp

g++ --std=c++17 -Wall -c -O3 search.cpp

g++ main.o search.o -o solution

root@Harry:~/work/MAI/DA/da\_lab4# g++ benchmark.cpp -o kmp

root@Harry:~/work/MAI/DA/da\_lab4# python3 testgen\_2.0.py

root@Harry:~/work/MAI/DA/da\_lab4# ./solution < test\_10^5.txt

Time: 105 microseconds

root@Harry:~/work/MAI/DA/da\_lab4# ./kmp < test\_10^5.txt

Time: 312 microseconds

root@Harry:~/work/MAI/DA/da\_lab4# python3 testgen\_2.0.py

root@Harry:~/work/MAI/DA/da\_lab4# ./solution < test\_10^6.txt

Time: 1695 microseconds

root@Harry:~/work/MAI/DA/da\_lab4# ./kmp < test\_10^6.txt

Time: 5930 microseconds

root@Harry:~/work/MAI/DA/da\_lab4# python3 testgen\_2.0.py

root@Harry:~/work/MAI/DA/da\_lab4# ./solution < test\_10^7.txt

Time: 16912 microseconds

root@Harry:~/work/MAI/DA/da\_lab4# ./kmp < test\_10^7.txt

Time: 30422 microseconds

Из тестов выяснено, что алгоритм построения Z-блоков с помощью Z-функции оказался быстрее алгоритма КМП примерно в 2-3 раза. Возможно, этому поспособствовало то, что в первом алгоритме считывались строки полные строки, а потом делились на отдельные слова, в то время как во втором алгоритме весь текст считывался посимвольно. Также, так как мы уже знаем, что сложность алгоритма построения Z-блоков является , то можно сделать вывод, что сложность алгоритма КМП тоже будет

# 6. Выводы

Была написана программа, в которой реализован поиск одного образца, основанный на построении Z-блоков. Выполняя лабораторную работе, я узнал, как работает Z-функция, и как применять её для поиска подстрок в тексте, которые совпадают с заданным образцом. Также были получены представления о том, как обрабатывать текст. Когда программа впервые была написана, она выполняла посимвольное чтение. В таком случае подсчёт слов в строке определялся с помощью подсчёта пробелов между словами и переноса строки. При отправке такой программы на чекер на 8 тесте выводилось сообщение о неверном ответе. Было предположено, что между словами в тексте и шаблоне может быть не один пробел. Поэтому считывание было переписано. Считывалась вся строка, после чего, она разделялась на слова, которые потом помещались в вектор слов. После того, как программа была переписана, она успешно прошла все тесты на чекере. После исправления программы были получены навыки для написания корректного считывания текста и разделения его на отдельные слова.

# Список используемых источников

1. Дэн Гасфилд. *Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология.* – Издательский дом «Невский Диалект, БХВ-Петербург», 2003. Перевод с английского И.В. Романовского. – 654 с. (ISBN 5-7940-0103-8 (“Невский Диалект”)).

2. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* – Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. – 1296 с. (ISBN 5-8459-0857-4 (рус.))

3. Роберт Лафоре. *Объектно-ориентированное программирование в C++*. Классика Computer Science, 4-е издание. — Издательский дом «Питер», 2018. Перевод с английского: А. Кузнецов, М. Назаров, В.Шрага. — 928 с. (ISBN 978-5-596-00353-7 (рус.))

4. *Информация о z-функции* – [Электронный ресурс]. – URL: <https://neerc.ifmo.ru/wiki/index.php?title=Z-функция> (дата обращения: 25.11.2020).

5. *Информация об алгоритмах поиска подстроки в строке* – [Электронный ресурс]. – URL: <http://algolist.ru/search/esearch/> (дата обращения: 25.11.2020).

6. *Информация о типе std::stringstream* – [Электронный ресурс]. – URL: <https://ru.cppreference.com/w/cpp/io/basic_stringstream> (дата обращения: 26.11.2020).

7. *Информация о библиотеке <chrono>* – [Электронный ресурс]. – URL: <https://ru.cppreference.com/w/cpp/chrono> (дата обращения: 27.11.2020).