Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Дискретный анализ»

Студент: Н. А. Абдыкалыков
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-23

Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №2

Задача: Требуется разработать программу, реализующую один из стандартных алгоритмов поиска образцов в тексте. Программа должна находить все вхождения заданного образца и выводить их местоположение в формате «номер строки, номер слова».

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

Вариант алфавита: `Слова не более 16 знаков латинского алфавита.`

1 Описание

Для решения поставленной задачи был реализован Z-алгоритм, так как он позволяет находить все вхождения образца в тексте за **линейное время** от суммарной длины образца и текста. Алгоритм работает с "алфавитом состоящим из слов, что точно соответствует условию.

Общая стратегия заключается в построении Z-массива для комбинированной строки вида `P$T`, где `P` — искомый образец, `T` — текст для поиска, а `$` — специальный символ-разделитель, который гарантированно не встречается ни в образце, ни в тексте. После вычисления Z-массива, вхождения образца легко находятся по тем индексам, где значение Z-функции равно длине образца.

Ключевым элементом является **Z-функция**. Для некоторой строки S её Z-массив Z определяется так, что $Z[i]$ — это длина наибольшего общего префикса строки S и её же суффикса, начинающегося с позиции i [1]. Вычисление Z-массива для комбинированной строки выполняется за один проход с использованием следующей оптимизации:

1. **Инициализация Z-значения.** Для каждой новой позиции i проверяется, не попадает ли она в границы $[l, r]$ самого правого из уже найденных Z-блоков. Если попадает, это позволяет инициализировать $Z[i]$ некоторым начальным значением, эффективно используя уже проделанную работу и избегая лишних сравнений.

2. **Расширение Z-блока.** После (или вместо) шага инициализации выполняется прямое "наивное"сравнение слов для максимально возможного расширения текущего Z-блока. Сравнение продолжается, пока слова из префикса комбинированной строки и из текущей позиции i совпадают.

3. **Обновление границ.** Если найденный для позиции i Z-блок простирается правее, чем текущий самый правый Z-блок (т.е. $i + Z[i] - 1 > r$), то границы l и r обновляются. Это позволяет использовать информацию о новом, более длинном Z-блоке на последующих итерациях.

2 Исходный код

Программа написана на языке C++ с использованием стандартной библиотеки. Основная логика вынесена в отдельные функции для улучшения читаемости и модульности.

На первой строке входного файла задается искомый образец. На последующих строках располагается текст, в котором осуществляется поиск. Для хранения слов текста и их исходных позиций была создана структура `TextData`. Весь ввод-вывод, а также непосредственно алгоритм поиска, реализованы с помощью набора функций, описанных в таблице ниже.

main.cpp	
`int main()`	Главная функция. Управляет процессом: вызывает функции чтения образца и текста, подготавливает данные для Z-алгоритма, запускает его и выводит найденные вхождения.
`TextData readText()`	Функция для считывания всех строк текста. Разбивает их на слова, приводит к нижнему регистру и сохраняет вместе с их координатами (номер строки и номер слова).
`vector<string> readPattern()`	Функция для считывания первой строки ввода, содержащей образец. Также разбивает ее на слова и приводит к нижнему регистру.
`vector<int> calculateZ(const vector<string>& s)`	Основная функция, реализующая Z-алгоритм. Вычисляет Z-массив для комбинированной последовательности слов `P$T`.
`void findAndPrintMatches(const vector<int>& zValues, int pLen, const vector<pair<int, int»& coords)`	Функция анализирует посчитанный Z-массив. Находит позиции, где значение Z-функции равно длине образца, и выводит соответствующие им координаты.
`string toLower(string s)`	Вспомогательная функция, которая преобразует строку в нижний регистр для обеспечения регистронезависимого поиска.

Структура для хранения данных о тексте определена следующим образом:

```
#include <string>
#include <vector>
#include <utility> // for std::pair

struct TextData {
    std::vector<std::string> words;
    std::vector<std::pair<int, int>> coordinates;
};
```

3 Консоль

Демонстрация работы программы: компиляция исходного кода, просмотр содержимого тестового файла `input.txt` и запуск исполняемого файла с перенаправлением ввода для поиска образца в тексте.

```
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ g++ -std=c++17 -
O2 -o program main.cpp
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ cat input.txt
cat dog cat dog bird
CAT dog CaT Dog Cat DOG bird CAT
dog cat dog bird
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ ./program < input.txt
1,3
1,8
```

4 Тест производительности

Для оценки эффективности реализованного Z-ал-ма было проведено сравнение его производительности с наивной реализацией поиска "в лоб". Асимптотическая сложность Z-ал-ма составляет $O(M + N)$, где M и N — длины образца и текста в словах, что является линейной сложностью. Наивный поиск в худшем случае имеет сложность $O(M \cdot N)$, что значительно медленнее на больших объемах данных.

Тестирование проводилось на большом сгенерированном тексте, состоящем из 10,000,000 (10^7) слов. Текст был специально составлен так, чтобы создавать "трудные"условия для наивного ал-ма (большое количество частичных совпадений с образцом). Для замера времени использовалась библиотека `<chrono>`. Компиляция обоих вариантов производилась с флагом оптимизации `-O2`.

laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ g++ -O2 -std=c++17 main.cpp -o z_searcher
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ ./z_searcher < large_test.txt
Z-algorithm search time: 0.3175 seconds

laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ g++ -O2 -std=c++17 naive_versi
o naive_searcher
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ ./naive_searcher < large_test.txt
Naive (brute-force) search time: 45.8214 seconds

Как показывают результаты, Z-ал-м работает на несколько порядков быстрее (в данном случае, примерно в 140 раз), чем наивный поиск. Это объясняется тем, что Z-ал-м обрабатывает весь текст за один проход, эффективно используя информацию о уже проверенных участках (Z-блоки), чтобы избежать повторных сравнений. Наивный же подход на каждом шаге начинает проверку заново, выполняя огромное количество избыточной работы. Линейная сложность Z-ал-ма демонстрирует колоссальное преимущество перед квадратичной сложностью простого подхода, что делает его единственным рабочим решением для обработки больших текстов.

5 Выводы

Z-алгоритм является высокоэффективным методом поиска, демонстрируя линейную временную сложность, которая не зависит от структуры текста или количества частичных совпадений. Однако его область применения специализирована на точном поиске фиксированных образцов и не подходит для более сложных задач, таких как поиск с использованием регулярных выражений.

В ходе выполнения лабораторной работы был реализован Z-алгоритм для поиска одного образца в тексте, состоящем из слов. Ключевой задачей стало достижение линейной производительности, в отличие от наивного подхода с квадратичной сложностью. Это было достигнуто путем реализации основной оптимизации Z-алгоритма — использования границ самого правого Z-блока для избежания повторных сравнений. Этот опыт наглядно демонстрирует, как выбор правильной алгоритмической стратегии может кардинально снизить временные затраты, и знакомит с устройством одного из классических и эффективных алгоритмов обработки строк.

Также был составлен отчет при помощи системы TEX, которая позволяет автоматизировать процесс создания качественной технической документации.

Список литературы

[1] 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.))

[2] *Z-функция — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Z-функция` (дата обращения: 20.09.2025).

[3] *Алгоритмы поиска подстроки — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Алгоритмы_поиска_подстроки` (дата обращения: 20.09.2025).