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: О. С. Концебалов
Преподаватель: А. А. Никитин
Группа: М8О-209Б-22
Дата: 25.05.2024
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №4

Задача: Необходимо реализовать поиск одного образца в тексте с использованием алгоритма Z-блоков. Алфавит — строчные латинские буквы.

Формат ввода: На первой строке входного файла текст, на следующей — образец. Образец и текст помещаются в оперативной памяти.

Формат вывода: В выходной файл нужно вывести информацию о всех позициях текста, начиная с которых встретились вхождения образца. Выводить следует по одной позиции на строчке, нумерация позиций в тексте начинается с 0.

1 Описание

Алгоритм Z-блоков используется для строковой обработки и анализа строк. Он полезен для задач, связанных с поиском подстрок в строке, поиска количества вхождения образца в строку и шаблонов. Основная идея алгоритма заключается в вычислении массива Z, который для каждой позиции строки показывает длину наибольшего префикса строки, начинающегося с этой позиции, который одновременно является префиксом всей строки. Алгоритм вычисления Z-функции работает за линейное время O(n). Основная идея алгоритма заключается в использовании уже вычисленных значений массива Z для ускорения вычислений. [1].

2 Исходный код

1. **Ввод данных**:
   - Пользователь вводит строку текста и шаблон, который необходимо найти в тексте.
2. **Объединение строк**:
   - Создаётся новая строка, путем конкатенации шаблона, символа-разделителя ($), и исходного текста. Это необходимо для разделения шаблона и текста, чтобы гарантировать, что они не повлияют друг на друга при вычислении Z-функции.
3. **Вычисление Z-функции**:
   - Инициализируется массив Z, где $Z[i]$ указывает длину наибольшего префикса строки, начинающегося с позиции i, который одновременно является префиксом всей строки.
   - Начальные границы самого правого Z-блока $left$ и $right$ устанавливаются в 0.
4. **Основной цикл вычисления Z-функции**:
   - Для каждой позиции i от 1 до конца объединённой строки выполняются следующие шаги:
     - Оптимизация вычислений, если для i символа мы уже считали Z-функцию. Если i находится внутри текущего Z-блока ($i \leq rigth$), значение $Z[i]$ инициализируется как минимум из $right - i + 1$ и значения Z для симметричной позиции внутри блока. В этом случае мы можем использовать ранее вычисленные значения Z-функции для обновления.
     - Если текущий символ совпадает с символом в префиксе строки, значение $Z[i]$ увеличивается.
     - Если обновлённая позиция конца совпадающего префикса $(i + Z[i] - 1)$ выходит за пределы текущего Z-блока, границы $left$ и $right$ обновляются.

```
#include <iostream>
#include <string>
#include <vector>

void calculateZFunc(std::vector<int>& zArray, const std::string& str) {
    for (int i = 1; i != str.length(); ++i) {
        int left = 0;
```

```
8           int right = 0;
9
10          if (i <= right) {
11              zArray[i] = std::min(right - i + 1, zArray[i - left]);
12          }
13
14          while (i + zArray[i] < str.length() && str[zArray[i]] == str[i + zArray[i
                ]]) {
15              ++zArray[i];
16          }
17
18          if (i + zArray[i] - 1 > right) {
19              left = i;
20              right = i + zArray[i] - 1;
21          }
22      }
23  }
24
25  int main() {
26      std::string text;
27      std::string pattern;
28
29      std::cin >> text >> pattern;
30
31      std::string stringToFunction = pattern + "$" + text;
32      std::vector<int> zArray (stringToFunction.length());
33
34      calculateZFunc(zArray, stringToFunction);
35
36      size_t patternLength = pattern.length();
37      for (size_t i = patternLength; i != zArray.size(); ++i) {
38          if (zArray[i] == patternLength) {
39              std::cout << i - patternLength - 1 << std::endl;
40          }
41      }
42
43      return 0;
44  }
```

3 Консоль

```
Для вводных данных:
abacaba
ab

baronpipistron@BaronPIpistron:~/diskran$ g++ -Wall -o out run.cpp
```

```
baronpipistron@BaronPIpistron:~/diskran$ ./out <input.txt
0
4
```

4 Тест производительности

Тест производительности представляет из себя следующее: сложность алгоритма Z-блоков составляет O(n), поэтому будет проводить тест роста времени с линейным ростом объема входных данных. Строка для поиска будет размером 1, 2, 3, 4 миллиона символов.

```
baronpipistron@BaronPIpistron:~/diskran$ g++ -Wall -o out run.cpp
baronpipistron@BaronPIpistron:~/diskran$ ./out <1_million_str.txt
Benchmark
Z-function O(n) tests
/------------------------------------------------------------------------/
Time of work Z-function: 21548930
/------------------------------------------------------------------------/
baronpipistron@BaronPIpistron:~/diskran$ ./out <2_million_str.txt
Benchmark
Z-function O(n) tests
/------------------------------------------------------------------------/
Time of work Z-function: 43572901
/------------------------------------------------------------------------/
baronpipistron@BaronPIpistron:~/diskran$ ./out <3_million_str.txt
Benchmark
Z-function O(n) tests
/------------------------------------------------------------------------/
Time of work Z-function: 54393290
/------------------------------------------------------------------------/
baronpipistron@BaronPIpistron:~/diskran$ ./out <4_million_str.txt
Benchmark
Z-function O(n) tests
/------------------------------------------------------------------------/
Time of work Z-function: 79434201
/------------------------------------------------------------------------/
```

Мы можем наблюдать линейный рост времени выполнения алгоритма Z-блоков в среднем в 2 раза при увеличении объёма вводных данных в 2 раза, за исключением погрешностей, которые могут возникать из-за внешних факторов, таких как нагрузка на систему. В зависимости от текущей загрузки системы, других запущенных процессов и доступных ресурсов процессора и памяти, время выполнения программы может варьироваться.

5 Выводы

В данной лабораторной работе был реализован алгоритм Z-блоков, который позволяет эффективно находить длину наибольшего общего префикса строки и всех ее суффиксов. Реализация алгоритма основана на линейном проходе по строке и использовании предыдущих результатов для определения значений Z-блоков. Реализация позволяет достичь временной сложности O(n), где n - длина строки. В целом, реализация алгоритма Z-блоков является эффективной и может быть применена для решения различных задач, связанных с обработкой строковых данных. Алгоритм довольно прост в реализации, но недостаточно эффективен. Есть множество других алгоритмов поиска подстроки в строке, которые решают эту задачу гораздо быстрее (КМТ, Рабин-Карп и т.д.) или одновременно производят поиск сразу нескольких шаблонов (Ахо-Корасик).

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.))