Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Дискретный анализ»

Студент: Н. А. Рудаков
Преподаватель: Н. С. Михайлова
Группа: М8О-201Б-21

Дата: 17.05.2023
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №4

Задача:

Вариант №4-1

Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.

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

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

Запрещается реализовывать алгоритмы на алфавитах меньшей размерности, чем указано в задании.

Программа должна обрабатывать строки входного файла до его окончания. Каждая строка может иметь следующий формат:

Формат входных данных

Искомый образец задаётся на первой строке входного файла.

Затем следует текст, состоящий из слов или чисел, в котором нужно найти заданные образцы.

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

Формат результата

В выходной файл нужно вывести информацию о всех вхождениях искомых образцов в обрабатываемый текст: по одному вхождению на строку.

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 строк, занятых образцами).

Порядок следования вхождений образцов несущественен.

1 Описание

Требуется написать реализацию Z-функции для поиска подстроки в строке (образца в тексте).

Справка вики[1]: **Z-фуннкция от строки S** — массив $Z_1, \ldots, Z_n$ такой что Z_i равен длине наибольшего общего префикса начинающегося с позиции i суффикса строки S и самой строки S.

Рассмотрим алгоритм вычисления Z-функции за линейное время:

1. Назовём отрезком совпадения подстроку, совпадающую с префиксом S. Будем поддерживать координаты l и r самого правого отрезка совпадения. Пусть i - текущий индекс, для которого мы хотим вычислить $Z_i(S)$.
2. Если $i <= r$ - попали в отрезок совпадения, так как строки совпадают, то и Z-блоки для них по отдельности совпадают $\Rightarrow Z_i(S) = z_{i-l}$. Так как $i + Z_i(S)$ может быть за пределами отрезка совпадения, то нужно ограничить значение величиной $r - i + 1$.
3. $i > r$ - тривиальный алгоритм (прсто прикладываем паттер к тексту , каждый раз сдвигая его на один символ).
4. В конце обновляем отрезок совпадения, если $i + Z_i(S) > r$ (тривиальный алгортим вышел за отрезок совпадения): $l = i, r = i + Z_i(S) - 1$.

Теперь рассмотрим алгоритм поиска подстроки в строке с помощью Z-фукнкции:

1. Во избежании путаницы назовём одну строку **текстом** t, а другую - **образцом** p. Таким образом, задача заключается в том, чтобы найти все вхождения образца p в текст t.
2. Для решения этой задачи образуем строку $s = p + \# + t$, т.е. к образцу припишем текст через символ-разделитель (который не всечается нигде в самих строках). Так как у нас слова могут состоять из букв латинского алфавита, а в теексте могу встречаться цифры, то в кчестве символа-разделителя я буду использовать знак \$.
3. Поссчитаем для полученной строки Z-функцию. Тогда для любого i в отрезке $[0; length(t)-1]$ по соответствующему значению $z[i+length(p)+1]$ можно понять, вхлдит ли образец p в текст t, начиная с позиции i: если это значение Z-функции равно $length(p)$, то да, входит, иначе - нет.

Таким образом, асимптотика решения получилась $O(length(t) + length(p))$. Потребление памяти имеет ту же асимптотику.

2 Исходный код

Для того, чтобы реализовать данный алгоритм необходимо как-то хранить номер строки, номер слова в строке и само слово вместе. Вначале, я подумал, что можно так не делать и хранить отдельно эти 3 параметра, но потом у меня вышли трудности с аккуратной сборкой параметров при выводе, поэтому было принятно решение хранить вместе индекс, номер строки и номер слова. По индексу можно понять конец слова и начало другого слова, потому что индекс мы инкрементируем при наборе слова из букв.

На первой строке нам дается паттерн, вводим его и чистим от ненужных пробелов, а также заменяем заглавные буквы на прописные. Далее вводиться текст посимвольно, набираем слова. Если у нас возникают символы переноса строки или пробела, то мы отсекаем слово. Также в set data мы кладем индекс последней буквы, номер текущей строки и номер текущего слова.

После набора текста применяем z-функцию. Передаем в нее строку вида pattern + # + text. Получаем результат z-функции в виде вектора длин совпадений с патерном. Итерируемся по вектору и в случае, если текущей элемент (длина совпадения с паттерном) = длине паттерна, то мы нашли совпадающую с паттерном подстроку в тексте. До этого мы посчитали номер строки и номер слова в строке. При помощи бинарного поиска по индексу найдем эти два параметра и положим в другой сет, чтобы избежать повторения ответов. Далее выводим ответ.

Код программы:

```
#include <vector>
#include <set>
#include <iostream>


std::vector<int> z_function(std::string &s) {
    int n = (int) s.length();
    std::vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = std::min(r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}

```

```
20 std::string remove_spaces(std::string &sequence) {
21     std::string new_str;
22     for (char i: sequence)
23         if (i != ' ')
24             new_str += i;
25     return new_str;
26 }
27
28 void to_lower(std::string &sequence) {
29     for (char &c: sequence)
30         c = (char) tolower(c);
31 }
32
33
34 int main() {
35     std::string pattern;
36     std::getline(std::cin, pattern);
37     to_lower(pattern);
38     pattern = remove_spaces(pattern);
39     int n = (int) pattern.size();
40     std::string text;
41     std::set<std::vector<int>> data;
42
43     char c;
44     int line = 1;
45     int index = 0;
46     int nWord = 0;
47     std::string curWord;
48
49     while ((c = (char) getchar()) != EOF) {
50         if (c == '\n') {
51             ++nWord;
52             if (!curWord.empty()) {
53                 text += curWord;
54                 data.insert({index, line, nWord});
55                 curWord = "";
56             }
57             ++line;
58             nWord = 0;
59         } else if (c == ' ') {
60             if (!curWord.empty()) {
61                 ++nWord;
62                 text += curWord;
63                 data.insert({index, line, nWord});
64                 curWord = "";
65             }
66         } else {
67             curWord += (char) tolower(c);
68             index += 1;
```

```
69          }
70      }
71
72      std::string find_sub = pattern + '#' + text;
73      std::vector<int> res = z_function(find_sub);
74
75      std::set<std::pair<int, int>> ans;
76
77      for (int i = 0; i < (int) res.size(); ++i) {
78          if (res[i] == n) {
79              std::vector<int> tmp = *data.lower_bound({i - n, 1, 1});
80              ans.insert({tmp[1], tmp[2]});
81          }
82      }
83      for (auto &i: ans)
84          std::cout << i.first << ", " << i.second << '\n';
85      return 0;
86  }
```

3 Консоль

```
ruda@ruda-lp:~/CLionProjects/lab4$ cat test1
cat dog cat dog bird
CAT dog CaT Dog Cat DOG bird CAT
dog cat dog bird
ruda@ruda-lp:~/CLionProjects/lab4$ cat test2
a
a b c a

a
a a
ruda@ruda-lp:~/CLionProjects/lab4$ g++ main.cpp
ruda@ruda-lp:~/CLionProjects/lab4$ ./a.out < test1
1,3
1,8
ruda@ruda-lp:~/CLionProjects/lab4$ ./a.out < test2
1,1
1,4
3,1
4,1
4,2
```

4 Тест производительности

Тест производительности представляет из себя следующее: мы обрабатываем различные тексты и измеряем время подсчёта для них Z-функции.

Для того, чтобы замерить работу z-функции и find(), я сгенерировал тест размером 200000 строк, после чего запустил программу.

```
ruda@ruda-lp:~/CLionProjects/lab4$ python3 test_generator.py
ruda@ruda-lp:~/CLionProjects/lab4$ g++ main.cpp
ruda@ruda-lp:~/CLionProjects/lab4$ ./a.out  <test
z-function time: 3188 ms
find function time: 4159 ms
```

Видно, что z-функция работает быстрее на целую секунду.

5 Выводы

Выполнив четвертую лабораторную работу по курсу «Дискретный анализ», я изучил алгоритм z-функции поиска подстроки в строке, смог обработать большой текст и точно определить местоположения подстрок, совпадающих с паттерном.

На самом деле, до этой лабораторной работы я применял z-функцию в реализации своих проектов, но в этой лабораторной работе я обработал большой текст с множественными пробелами и пустыми строками. Это немного отличается от обычного применения z-функции, по крайней мере, в моем представлении.

Список литературы

[1] *АВЛ-дерево — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Z-функция` (дата обращения: 5.01.2021).