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: Л. В. Ивенкова
Преподаватель: Н. С. Капралов
Группа: М8О-208Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

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

В начале создаём структуру случайного слова **TWord**, в которой будут лежать строка, хранящая собственно само слово, и два числа - индекс строки, в которой находится этого слово, и его порядковый номер в ней.

Далее реализуем саму Z-функцию, а так же вспомогательную функцию перевода букв слов в сторочные символы (для реализации регистронезависимости нашей программы).

Потом в main посимвольно считываем сначала образец, а потом текст (посимвольное считывание нужно для правильной обработки знаков пробела, переноса строк, конца файла, а так же самих букв в словах). Парралельно с каждым знаком ' ' мы увеличиваем наш счётчик слов, а при переносе строки обнуляем его и увеличиваем счётчик строк. После считывания образца не забываем добавить к результирующей строке s символ \$.

Сам код:

```
#include <iostream>
#include <cstring>
#include <vector>
const unsigned int WORD_SIZE = 17;

struct TWord {
    std::string Word = "";
    unsigned int WordIndex;
    unsigned int LineIndex;

    TWord() = default;
    TWord(std::string &s, unsigned int wi, unsigned int li) {
        Word = s;
        WordIndex = wi;
        LineIndex = li;
    };
    ~TWord() = default;

    const bool operator==(const TWord &rhs) const {
        for (unsigned int i = 0; i < WORD_SIZE; ++i)
            if (Word[i] != rhs.Word[i])
                return false;
            else return true;
    }
};
```

```
26
27  std::vector<unsigned int> ZFunction(const std::vector<TWord> &text) {
28      unsigned int TextSize = text.size();
29      unsigned int left = 0;
30      unsigned int right = 0;
31      std::vector<unsigned int> result(TextSize);
32      for (unsigned int i = 1; i < TextSize; ++i) {
33          if (i <= right) {
34              result[i] = std::min(result[i-left], right - i);
35          }
36          while ((result[i] + i < TextSize) && (text[result[i]].Word == text[result[i] +
                i].Word)) {
37              ++result[i];
38          }
39          if (i + result[i] > right) {
40              left = i;
41              right = result[i] + i;
42          }
43      }
44      return result;
45  }
46
47  void ToLower(char* str) {
48      if (*str>='A' and *str<='Z')
49          *str -= 'A'-'a';
50  }
51
52  int main() {
53
54      char c;
55      std::string word = "";
56      unsigned int PatternSize = 0;
57      unsigned int WordNumber = 0;
58      unsigned int LineNumber = 1;
59      std::vector<TWord> text;
60
61      while ((c = getchar()) != EOF) {
62          if (c == '\n') {
63              if (word != "") {
64                  ++WordNumber;
65                  text.push_back(TWord(word,WordNumber,0));
66                  ++PatternSize;
67                  word = "";
68              }
69              break;
70          }
71          else if (c == ' ') {
72              if (word != "") {
73                  ++WordNumber;
```

```
74                  text.push_back(TWord(word,WordNumber,0));
75                  ++PatternSize;
76                  word = "";
77              }
78          }
79          else {
80              ToLower(&c);
81              word += c;
82          }
83      }
84
85      if (PatternSize == 0) return 0;
86
87      std::string x = std::string("$");
88      text.push_back(TWord(x,0,0));
89
90      WordNumber = 0;
91
92      while ((c = getchar()) != EOF) {
93          if (c == '\n') {
94              ++WordNumber;
95              if (word != "") {
96                  text.push_back(TWord(word,WordNumber,LineNumber));
97                  word = "";
98              }
99              ++LineNumber;
100             WordNumber = 0;
101         }
102         else if (c == ' ') {
103             if (word != "") {
104                 ++WordNumber;
105                 text.push_back(TWord(word,WordNumber,LineNumber));
106                 word = "";
107             }
108         }
109         else {
110             ToLower(&c);
111             word +=c;
112         }
113     }
114
115     std::vector<unsigned int> Result = ZFunction(text);
116
117     for (int i = 0; i < text.size(); i++) {
118         if (Result[i] == PatternSize) std::cout << text[i].LineIndex << ", " << text[i
                ].WordIndex << "\n";
119     }
120     return 0;
121 }
```

3 Консоль

```
parsifal@DESKTOP-3G70RV4:~/DA/Lab4$ python3 generator.py
Pattern =  ['dog','cat','dog','dog','bird']
parsifal@DESKTOP-3G70RV4:~/DA/Lab4$ cat test01.t
dog cat dog dog bird
doctor parzival Yatty AttackHelicopter parzival cat
dog dog
cat
dog dog bird
doctor AttackHelicopter parzival doctor Yatty parzival dog cat
dog dog bird
parsifal@DESKTOP-3G70RV4:~/DA/Lab4$ g++ main.cpp
parsifal@DESKTOP-3G70RV4:~/DA/Lab4$ ./a.out  <test01.t
2,2
5,7
```

4 Тест производительности

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

Для этого создаём программу для генерации тестов.

```
1  import random
2  import string
3  from random import choice
4
5  def main():
6      pattern_size = random.randint(3, 10)
7      pattern = []
8      text = []
9      words_in_pattern = ["dog", "cat", "bird"]
10     words_in_text = ["dog", "cat", "bird", "Who", "Yatty", "AttackHelicopter", "doctor"
           , "parzival", "Nyaaa", "tyan"]
11     for i in range(pattern_size):
12         pattern.append(choice(words_in_pattern))
13     count = 0
14     while (count < 10000):
15         rand = random.randint(1, 100)
16         if (rand > 90):
17             for i in pattern:
18                 text.append(i)
19             count = count + pattern_size
20         else:
21             text.append(choice(words_in_text))
22             count = count + 1
23
24     test_file = open('test01.t', 'w')
25     for i in range(len(pattern)):
26         if (i == len(pattern) - 1):
27             test_file.write(pattern[i] + '\n')
28         else:
29             test_file.write(pattern[i] + ' ')
30
31     # test_file.write('\n')
32     for i in text:
33         rand = random.randint(1, 100)
34         if (rand > 85):
35             test_file.write('\n')
36         test_file.write(i + ' ')
37
38     test_file.close()
39
```

```
40      print('Pattern = ', pattern)
41  main()
```

Так же создаём бенчмарк, для измерения времени работы Z-функции (с помощью библиотеки chrono - она позволяет замерить время от запуска функции до её завершения). Замерям время работы двух Z-функций - одной, основанной на алгоритме линейной сложности, и второй - на наивном алгоритме квадратичной сложности. Причём проверям на одинаковых текстах, но разных рамеров.

```
parsifal@DESKTOP-3G70RV4:~/DA/Lab4$ python3 generator.py
Pattern =  ['cat','bird','cat','cat','dog','dog','bird','dog']
parsifal@DESKTOP-3G70RV4:~/DA/Lab4$ g++ benchmark.cpp
parsifal@DESKTOP-3G70RV4:~/DA/Lab4$ ./a.out  <test01.t
5815 вхождений
Time O(n): 3916
Time O(n^2): 5344
```

Количество слов в тексте	Время работы
1000	Наивного алгоритма: 7 Линейного алгоритма: 7
1000	Наивного алгоритма: 57 Линейного алгоритма: 49
10.000	Наивного алгоритма: 535 Линейного алгоритма: 444
100.000	Наивного алгоритма: 6115 Линейного алгоритма: 4878
1.000.000	Наивного алгоритма: 49608 Линейного алгоритма: 34142
10.000.000	Наивного алгоритма: 534655 Линейного алгоритма: 419207

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я изучила различные алгоритмы поиска подстроки в строке, а так же научилась реализовывать их несколькими способами. Смогла с их помощью обрабатывать не только строки, но и болшие тексты.

Научилась генерировать тестовые тексты для этих поисков, а также оценивать сложность и эффективность их работы.

Список литературы

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