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: К. М. Воронов
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №4

Задача: Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита. Запрещается реализовывать алгоритмы на алфавитах меньшей размерности, чем указано в задании.

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

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

1 Описание

Требуется написать реализацию Z-функции.

Пусть дана строка s длины n. Как сказано в [1]: «Z-функция от этой строки — это массив длины n, i-ый элемент которого равен наибольшему числу символов, начиная с позиции i, совпадающих с первыми символами строки s.». Нулевая позиция Z-функции не заполняется, так как строка совпадает сама с собой и заполнять её длиной строки не имеет смысла.

Существует два алгоритма построения Z-функции: наивный - O(n^2), и эффективный - O(n). Суть навиного заключается в сравнивании элементов, начиная с нуля, для каждого позиции i. Эффективный же использует уже посчитанные данные. Создадим две переменные, left и right, изначально равные нулю. Они будут обозначать самый правый отрезок, на котором мы проходились наивным алгоритмом, и если мы находимся в пределах этого отрезка, это значит, что мы возможно уже можем записать в z[i] не 0, а большее число. Для этого нам надо посмотреть в z[i - left] (или же right - i, если это значение больше, чем осталось до конца строки) и продолжить наивный алгоритм с позиции i + z[i], не забывая при этом поменять left и right при сдвиге отрезка ещё правее. Если же мы находимся за границей, то есть правее right, мы также идём наивным алгоритмом и сдвигаем отрезок.

2 Исходный код

Для хранения слов последовательности создадим структуру TWord с тремя полями: строкой Word, в которой и будет хранится слово, номером строки этого слова NumberString и номером слова в этой строке NumberWord. После считывания строки создадим и заполним Z-функцию (искомый образец + $ + текст) эффективным алгоритмом, после чего пройдемся и выведем все вхождения.

```
#include<iostream>
#include <cstdio>
#include <vector>
#include <locale>
#include <algorithm>
#include <cstring>
#include <string>

using namespace std;

struct TWord {

    string Word;
    unsigned long long NumberWord;
    unsigned long long NumberString;

    TWord (string c, unsigned long long nw, unsigned long long ns) {
        Word = c;
        NumberWord = nw;
        NumberString = ns;
    }

    bool operator == (const TWord &a) {

        if (Word.size() != a.Word.size()) {
            return false;
        }

        for (int i = 0; i < Word.size(); ++ i) {
            if (a.Word[i] != Word[i]) {
                return false;
            }
        }
        return true;
    }

    char operator[] (unsigned long long i) {
        return Word[i];
    }

};
```

```
42
43  void ToLower (char &c) {
44      if ('A' <= c && c <= 'Z') {
45          c = c - 'A' + 'a';
46      }
47  }
48
49  void Z(vector<unsigned long long> &z, vector<TWord> &patternText) {
50
51      unsigned long long left = 0, right = 0;
52      unsigned long long count = 0;
53      unsigned long long j;
54
55      for (unsigned long long i = 1; i < patternText.size(); ++i) {
56
57          j = i;
58          if (i <= right) {
59              z[i] = min(z[i - left], right - i);
60              j = i + z[i];
61              count += z[i];
62          }
63
64          while ((j < patternText.size()) && (patternText[count] == patternText[j])) {
65              j += 1;
66              count += 1;
67          }
68
69          z[i] = count;
70          if (j >= right) {
71              left = i;
72              right = j;
73          }
74          count = 0;
75      }
76  }
77
78  int main() {
79      vector<TWord> patternText;
80      char c;
81      string c2 = "";
82      unsigned long long numberWord = 1, numberString = 0;
83      unsigned long long sizeWord = 0;
84      int first = 0;
85
86      while ((c = getchar()) != EOF) {
87
88          ToLower(c);
89
90          if (c == ' ' && c2 == "") {
```

```
91              continue;
92          }
93
94          if (c == '\n' && c2 == "") {
95              if (first == 0) {
96                  first = 1;
97                  c2 = "$";
98                  patternText.push_back({c2, 0, 0});
99                  c2 = "";
100             }
101             numberString += 1;
102             numberWord = 1;
103             continue;
104         }
105
106         if ((c == ' ') && (c2 != "")) {
107             patternText.push_back({c2, numberWord, numberString});
108             c2 = "";
109             if (first == 0) {
110                 sizeWord += 1;
111             } else {
112                 numberWord += 1;
113             }
114         }
115
116         if ((c == '\n') && (c2 != "")) {
117             if (first == 0) {
118                 patternText.push_back({c2, 0, 0});
119                 c2 = "";
120                 sizeWord += 1;
121                 c2 = "$";
122                 patternText.push_back({c2, 0, 0});
123                 c2 = "";
124                 first = 1;
125                 numberString += 1;
126             } else {
127                 patternText.push_back({c2, numberWord, numberString});
128                 c2 = "";
129                 numberString += 1;
130                 numberWord = 1;
131             }
132         }
133
134         if ((c != EOF) && (c != '\n') && (c != ' ')) {
135             c2 += c;
136         }
137     }
138
139     if (c2 != "") {
```

```
140         patternText.push_back({c2, numberWord, numberString});
141         c2 = "";
142     }
143 
144     if (sizeWord == 0) {
145         return 0;
146     }
147 
148     if (patternText.size() == sizeWord + 1) {
149         return 0;
150     }
151 
152     vector<unsigned long long> z(patternText.size());
153     Z(z,patternText);
154 
155     for (unsigned long long i = sizeWord + 1; i < patternText.size() ; ++i) {
156         if (z[i] == sizeWord) {
157             printf("%llu, %llu\n", (patternText[i]).NumberString, (patternText[i]).
                    NumberWord);
158         }
159     }
160 
161 }
```

3 Консоль

```
kirill@kirill-VirtualBox:~/Рабочий стол/DA/lab4$ cat test
abab
ababc abab cbab cbabc bbb
aa ababcabcbd abab
kirill@kirill-VirtualBox:~/Рабочий стол/DA/lab4$ cat test | ./s*
1,2
2,3
```

4 Тест производительности

Тест производительности представляет из себя следующее: сравнение наивного и эффективного алгоритмов построения . Текст состоит из 10^5 и 10^6, а образцы 10^3 и 10^4 соответственно.

```
kirill@kirill-VirtualBox:~/Рабочий стол/DA/lab4$ cat ./tests/test02.txt | ./s*
Время работы эффективного алгоритма: 9.067 ms
Время работы наивного алгоритма: 14.347 ms

kirill@kirill-VirtualBox:~/Рабочий стол/DA/lab4$ cat ./tests/test01.txt | ./s*
Время работы эффективного алгоритма: 153.171 ms
Время работы наивного алгоритма: 295.839 ms
```

Как видно, эффективный алгоритм работает быстрее.

5 Выводы

Выполнив четвёртую лабораторную работу по курсу «Дискретный анализ», я изучил работу алгоритмов поиска подстроки в строке, написал наивный и эффективный алгоритмы построения Z-функции и сравнил их. Также я задумался над тем, как такие алгоритмы придумывались, какой ход мыслей был у их создателей.

Список литературы

[1] Z-функция строки и её вычисление.
`https://e-maxx.ru/algo/z_function` (дата обращения: 08.12.2020).

[2] Лекции Н.К.Макарова, Московский Авиационный Институт.