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: Т. А. Габдуллин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2019

Лабораторная работа №4

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

Вариант алгоритма: 4. Поиск одного образца основанный на построении Z-блоков

Вариант алфавита: 1. Слова не более 16 знаков латинского алфавита (регистронезависимые).

Входные данные: Искомый образец задается на первой строке входного файла. В случае, если в задании требуется найти несколько образцов, они задаются по одному на строку вплоть до пустой строки.

Затем следует текст, состоящий из слов или чисел, в котором нужно найти заданные образцы. Никаких ограничений на длину строк, равно как на количество слов или числ в них, не накладывается.

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

1 Метод решения

Пусть дана строка s длины n. Тогда Z-функция ("зет-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.

Иными словами, $z[i]$ — это наибольший общий префикс строки s и её i-го суффикса.

Первый элемент Z - функции, $z[0]$, обычно считают неопределённым. В данной статье мы будем считать, что он равен нулю (хотя ни в алгоритме, ни в приведённой реализации это ничего не меняет).

Чтобы получить эффективный алгоритм, будем вычислять значения $z[i]$ по очереди — от $i = 1$ до $n - 1$, и при этом постараемся при вычислении очередного значения $z[i]$ максимально использовать уже вычисленные значения.

Назовём для краткости подстроку, совпадающую с префиксом строки s, отрезком совпадения. Например, значение искомой Z-функции $z[i]$ — это длиннейший отрезок совпадения, начинающийся в позиции i (и заканчиваться он будет в позиции $i+z[i]-1$).

Для этого будем поддерживать координаты $[l; r]$ самого правого отрезка совпадения, т.е. из всех обнаруженных отрезков будем хранить тот, который оканчивается правее всего. В некотором смысле, индекс r — это такая граница, до которой наша строка уже была просканирована алгоритмом, а всё остальное — пока ещё не известно.
Линейная оценка: $\emptyset(n)$
Описание взято с сайта *e-maxx.ru*.

2 Листинг

1. main.cpp

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

using namespace std;

vector<string> ReadPattern();
void BuildZBlocks(vector<string> pattern, vector<size_t> &z);
void ZFunc(vector<string> &pattern);
void CalculateSP(const vector<size_t> &z, vector<int> &spWeak);
void FailureFunc(vector<int> &sp, vector<int> &fail);

int main() {

    vector<string> pattern = ReadPattern();
    ZFunc(pattern);

    return 0;
}

void ZFunc(vector<string> &pattern) {

    size_t pattSize = pattern.size();

    vector<size_t> z(pattSize);
    BuildZBlocks(pattern, z);

    vector<int> sp(pattSize);
    CalculateSP(z, sp);
    z.clear();

    vector<int> fail(pattSize + 1);
    FailureFunc(sp, fail);
    sp.clear();

    char ch;
    string word;
    bool wordFound = false;

    queue<pair<int, int> > pos;
    pair<int, int> tmp;
    tmp.first = 1;
    tmp.second = 0;

```

```
47      size_t pattPos = 0;
48
49      do {
50          ch = getchar();
51          if (ch == ' ' || ch == '\t' || ch == EOF) {
52              if (wordFound) {
53                  ++tmp.second;
54                  if (pos.size() == pattSize) {
55                      pos.pop();
56                  }
57                  pos.push(tmp);
58
59                  while (pattPos > 0 && pattern[pattPos] != word) {
60                      pattPos = fail[pattPos];
61                  }
62
63                  if (pattern[pattPos] == word) {
64                      ++pattPos;
65                  }
66                  if (pattPos == pattSize) {
67                      //
68                      cout << pos.front().first << ", " << pos.front().second << endl
                            ;
69                      pattPos = fail[pattPos];
70                  }
71              }
72              wordFound = false;
73          }
74          else if (ch == '\n' ) {
75              if (wordFound) {
76                  ++tmp.second;
77                  if (pos.size() == pattSize) {
78                      pos.pop();
79                  }
80                  pos.push(tmp);
81
82                  while (pattPos > 0 && pattern[pattPos] != word) {
83                      pattPos = fail[pattPos];
84                  }
85                  if (pattern[pattPos] == word) {
86                      ++pattPos;
87                  }
88                  if (pattPos == pattSize) {
89                      //
90                      cout << pos.front().first << ", " << pos.front().second << endl
                            ;
91                      pattPos = fail[pattPos];
92                  }
93              }
```

```
94              ++tmp.first;
95              tmp.second = 0;
96              wordFound = false;
97          }
98          else {
99              wordFound = true;
100             word += tolower(ch);
101             continue;
102         }
103         word.clear();
104     } while (ch != EOF);
105 }
106
107 void BuildZBlocks(vector<string> pattern, vector<size_t> &z) {
108
109     size_t pattSize = pattern.size();
110     size_t r = 0;
111     size_t l = 0;
112
113     for (size_t i = 1; i < pattSize; ++i) {
114         if (i < r) {
115             z[i] = min(r - i + 1, z[i - l]);
116         }
117         while (i + z[i] < pattSize && pattern[z[i]] == pattern[i + z[i]])
118             ++z[i];
119         if (i + z[i] - 1 > r)
120             l = i;
121             r = i + z[i] - 1;
122     }
123 }
124
125 void CalculateSP(const vector<size_t> &z, vector<int> &spWeak) {
126
127     int size = z.size();
128     vector<int> spStrict(size);
129     int i;
130
131     for (int j = size - 1; j > 0; j--) {
132         i = j + z[j] - 1;
133         spStrict[i] = z[j];
134     }
135
136     spWeak[size - 1] = spStrict[size - 1];
137     for (i = size - 2; i > 0; i--) {
138         spWeak[i] = max(spWeak[i + 1] - 1, spStrict[i]);
139     }
140 }
141
142 void FailureFunc(vector<int> &sp, vector<int> &fail) {
```

```
143 
144     int size = fail.size();
145     for (size_t i = 1; i < size; i++) {
146         fail[i] = sp[i - 1];
147     }
148 
149 // cout << "FailFunc: " << endl;
150 // for (int i = 0; i <= size; i++) {
151 // cout << i << ": " << fail[i] << endl;
152 // }
153 }
154 
155 vector<string> ReadPattern() {
156 
157     string word;
158     char ch;
159     vector<string> pattern;
160 
161     do {
162         ch = getchar();
163         if (ch == ' ' || ch == '\n') {
164             if (!word.empty()) {
165                 pattern.push_back(word);
166             }
167             word.clear();
168         } else {
169             word += tolower(ch);
170         }
171     } while (ch != '\n');
172 
173 
174     return pattern;
175 }
```

3 Выводы

Благодаря данной лабораторной работе, я узнал о алгоритме поиски подстроки в строке, который основан на Z функциях - это массив длины n, i-ый элемент которого равен наибольшему числу символов, начиная с позиции i, совпадающих с первыми символами строки s , иными словами $z[i]$ - это наибольший общий префикс строки s и её i-го суффикса
Так же я познакомился с тривиальным алгоритмом, где для каждой позиции i перебираем ответ для неё $z[i]$, начиная с нуля, и до тех пор, пока мы не обнаружим несовпадение или не дойдём до конца строки. Разумеется, эта реализация слишком неэффективна, и ее временная оценка равна $\emptyset(n^2)$.
А так же есть эффективный алгоритм вычисления Z функций, описанный выше, который работает аж в n раз быстрее!

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-функции*
URL:`http://www.e-maxx-ru.1gb.ru/algo/z`$_{function}$