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. А. Калугин
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №4

Задача: Требуется разработать программу, осуществляющую ввод паттерна и текста с алфавитом, состоящем из слов, длиной не более 16 букв. Каждое слово состоит из латинских регистронезависимых букв. Программа считывает паттерн и текст, после чего производит поиск подстроки (паттерна) в строке (тексте) при помощи алгоритма Апостолико-Джанкарло[1]. Результаты поиска выводятся.

Вариант алгоритма: Поиск одного образца при помощи алгоритма Апостолико-Джанкарло.

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

1 Описание

Суть алгоритма Апостолико-Джанкарло заключается в добавлении к обычному алгоритму Бойера-Мура 5 специальных правил, которые позволяют пропускать большое количество сравнений[1]. Таким образом, нам необходимо реализовать функции определения длины сдвига - правило плохого символа, правило хорошего суффикса и правило сдвига при полном совпадении, после чего добавить правила, по которым выполняется сравнение подстроки со строкой.

2 Исходный код

Для начала производиться считывание паттерна и текста. Это осуществляется при помощи 2 циклов while. Далее выполняется препроцессинг - считается Z-функция от реверсированного патерна, на ёе основании строится массив N, инициализируются массивы L и M. L также заполняется данными. Определяется l - длина максимального возможного сдвига при полном совпадении. Создается и заполняется map «bsv» - она используется для правила плохого символа. На этом препроцессинг заканчивается. Сам же алгоритм поиска состоит из 2 вложенных циклов while. Первый проходит от первого до $(|t| - |p|)$-го элемента (где $|t|$ и $|p|$ - это длины текста и паттерна соответсвенно) и "прикладывает"паттерн к тексту для сравнения. Это происходит во втором цикле, который осуществляет так называемую "фазу сравнения". Она осуществлена при помощи пяти правил Апостолико-Джанкарло. Далее, в зависимости от того, было ли найдено полное совпадение происходит либо вывод ответа (если было) и сдвиг на $|p| - l$ элементов, либо (в противном случае) просто сдвиг на максимальное из трех (1, длина сдвига по правилу плохого символа, длина сдвига по правилу хорошего символа) значение. После этого внешний while-цикл повторяется.

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <map>
#include <ctime>

using namespace std;

vector <int> Zfunc (vector <string> input) {
    vector <int> Z;
    for (int i = 0; i < int (input.size ()); i ++) {
        Z.push_back (0);
    }
    int count;
    int l = 0;
    int r = 0;
    for (int i = 1; i < int (input.size ()); i ++) {
        count = 0;
        int k = i;
        int j = 0;
        if ((l <= i) && (i <= r)) {
            if (r - i >= Z [i - l]) {
                Z [i] = Z [i - l];
            }
            else {
                l = k;
                count += r - k + 1;
                j += r - k + 1;
```

```
30                k = r + 1;
31
32                if ((k < int (input.size ())) && (input [k] == input [j])) {
33                    l = i;
34                }
35
36                while ((k < int (input.size ())) && (input [k] == input [j])) {
37                    r = k;
38                    k ++;
39                    j ++;
40                    count ++;
41                }
42            }
43        }
44        else {
45            if ((k < int (input.size ())) && (input [k] == input [j])) {
46                l = i;
47            }
48
49            while ((k < int (input.size ())) && (input [k] == input [j])) {
50                r = k;
51                k ++;
52                j ++;
53                count ++;
54            }
55        }
56        Z [i] = count;
57    }
58    return Z;
59 }
60
61 struct TPos {
62    int line= 1;
63    int col = 1;
64 };
65
66 int main () {
67    vector <string> p;
68    vector <string> pr;
69    vector <string> t;
70    vector <int> N;
71    vector <int> L;
72    vector <int> M;
73    vector <TPos> place;
74    TPos pl;
75    int j1 = 0;
76    string s;
77    int l = 0;
78    char c = 1;
```

```
79      bool f = false;
80      int plus = 1;
81
82      while (c > 0) {
83          c = getchar();
84
85          if (c >= 65 && c <= 90) {
86              s += c;
87              f = true;
88          }
89
90          if (c >= 97 && c <= 122) {
91              c -= 32;
92              s += c;
93              f = true;
94          }
95
96          if (f && ((c == '\t') || (c == ' '))) {
97              p.push_back (s);
98              s = "";
99              f = false;
100         }
101
102         if (f && ((c == '\n') || (c == EOF))) {
103             p.push_back (s);
104             s = "";
105             f = false;
106             break;
107         }
108
109         if (!f && ((c == '\n') || (c == EOF))) {
110             break;
111         }
112     }
113
114     f = false;
115     if (int (p.size ()) > 0) {
116     while (c > 0) {
117         c = getchar ();
118         if (c == '\n') {
119             if (f) {
120                 t.push_back (s);
121                 place.push_back (pl);
122                 s = "";
123                 f = false;
124             }
125             pl.line++;
126             pl.col = 1;
127         }
```

```
128
129         if (c == EOF) {
130             if (f) {
131                 t.push_back (s);
132                 place.push_back (pl);
133                 s = "";
134                 f = false;
135             }
136             pl.line++;
137             pl.col = 1;
138             break;
139         }
140
141         if ((c == ' ') || (c == '\t')) {
142             if (f) {
143                 t.push_back (s);
144                 place.push_back (pl);
145                 s = "";
146                 f = false;
147                 pl.col ++;
148             }
149
150         }
151
152         if (c >= 65 && c <= 90) {
153             s += c;
154             f = true;
155         }
156
157         if (c >= 97 && c <= 122) {
158             c -= 32;
159             s += c;
160             f = true;
161         }
162     }
163     if (t.size () >= p.size ()) {
164
165     f = false;
166     pr = p;
167     reverse (pr.begin (), pr.end ());
168     N = Zfunc (pr);
169     reverse (N.begin (), N.end ());
170
171     for (int i = 0; i < int (N.size ()); i ++) {
172         L.push_back (-1);
173     }
174
175     for (int i = 0; i < int (t.size ()); i ++) {
176         M.push_back (-1);
```

```
177     }
178
179     for (int i = 0; i < int (N.size ()); i ++) {
180         int j = N.size () - N [i];
181         if (j < int (N.size ())) {
182             L [j] = i;
183         }
184     }
185
186     for (int i = int (p.size ()); i > 0; i --) {
187         int j = int (p.size ()) - i + 1;
188         if (N [j - 1] == j) {
189             l = max (j, l);
190         }
191     }
192
193     map <string, int> bsv;
194     for (int i = 0; i < int (p.size ()); i++) {
195         bsv[p [i]] = i;
196     }
197
198     int i = 0;
199         while (i <= int (t.size ()) - int (p.size ())) {
200
201             int j = int (p.size ()) - 1;
202             int h = i + j;
203
204             while (1) {
205                 if ((M [h] == -1) || (M [h] == 0)) {//1
206                     if ((t [h] == p [j]) && (j == 0)) {
207                         f = true;
208                         M [i + int (p.size ()) - 1] = int (p.size ());
209                         break;
210                     }
211                     else if ((t [h] == p [j]) && (j > 0)) {
212                            h --;
213                            j --;
214                     }
215                     else if (t [h] != p [j]) {
216                         M [i + int (p.size ()) - 1] = i + int (p.size ()) - 1 - h;
217                         f = false;
218                         j1 = j;
219                         break;
220                     }
221                 }
222                 else if (M [h] < N [j]) {//2
223                     j -= M [h];
224                     h -= M [h];
225                 }
```

```
            else if ((M [h] >= N [j]) && (N [j] == j + 1) && (j + 1 > 0)) {//3
                f = true;
                M [i + int (p.size ()) - 1] = i + int (p.size ()) - 1 - h;
                break;
            }
            else if ((M [h] > N [j]) && (N [j] < j + 1)) {//4
                M [i + int (p.size ()) - 1] = i + int (p.size ()) - 1 - h;
                f = false;
                j1 = j - N [j];
                break;
            }
            else if ((M [h] == N [j]) && (0 <= N [j]) && (N [j] < j + 1)) {//5
                j -= M [h];
                h -= M [h];
            }
            else {
                break;
            }
        }

        if (f) {
            cout << place [i].line<< ", "<< place [i].col << endl;
            i += int (p.size ()) - l;
        } else {
            if (bsv.count (t [j1 + i]) > 0) {
                int x = (*bsv.find (t [j1 + i])).second;
                if (x > 1) {
                    plus = max (j1 - x, plus);
                }
            }
            else {
                plus = max (j1 + 1, plus);
            }

            if ((j1 + 1 >= int (p.size ())) || ((L [j1 + 1]) == -1)) {
                plus = max (1, plus);
            } else {
                plus = max ((int (p.size ()) - 1 - L [j1 + 1]), plus);
            }

            plus = max (1, plus);
            if (i + plus + int (p.size ()) <= int (t.size ())) {
                i += plus;
            }
            else {
                break;
            }
            plus = 1;
        }
```

```
275             f = false;
276         }
277     }
278     }
279     }
280 return 0;
281 }
```

1.cpp	
struct TPos { int line= 1; int col = 1; };	Класс для хранения номеров позиций в тексте
vector <int> Zfunc (vector <string> input)	Функция подсчета Z-функции

3 Консоль

```
PS C:\Users\Кирилл\Desktop\VSC\DA>Get-Content .\test.txt
cat dog cat dog bird
CAT dog CaT Dog Cat DOG bird CAT
dog cat dog bird
PS C:\Users\Кирилл\Desktop\VSC\DA>Get-Content .\test.txt | .\a.exe
1,3
1,8
```

4 Тест производительности

В тесте производительности сравнивается быстродействие наивного алгоритма поиска подстроки в строке и алгоритм Апостолико-Джанкарло. Тесты представляют из себя паттерн и текст, состоящие из 1000 и 1000000, 1000 и 10000000, 10000 и 10000000 букв "a".

```
PS C:\Users\Кирилл\Desktop\VSC\DA>.\test.exe|.\eff.exe;.\test.exe|.\neff.exe
Time: 0.229 sec
Time: 64.366 sec

PS C:\Users\Кирилл\Desktop\VSC\DA>.\test.exe|.\eff.exe;.\test.exe|.\neff.exe
Time: 1.587 sec
Time: 601.702 sec

PS C:\Users\Кирилл\Desktop\VSC\DA>.\test.exe|.\eff.exe;.\test.exe|.\neff.exe
Time: 1.436 sec
Time: 4657.65 sec
```

Как видно, время работы наивного алгоритма поиска несоизмеримо больше. Это связано с тем, что ckj;yjcnm работы алгоритма Апостолико-Джанкарло - линейное, а наивного - O($M * N$). Таким образом, на максимальном тесте наивный алгоритм делает 10^{11} сравнений, что и вызывает время работы в 77 минут.

5 Выводы

Выполнив эту работу я научился нескольким вещам.
Во-первых, при написании программы необходимо учитывать, что она может быть запущена на разных ОС. В моем случае это приводило к тому, что корректно работающее на Windows считывание данных, не работало на Linux из-за разных символов конца файла (\n в Windows и EOF в Linux).
Во-вторых, программа на C++, даже написанная с грубыми ошибками (например вылетом за границы массива) может проходить некоторые тесты из-за особенностей работы с памятью в этом языке (А именно - возможности использования "мусорных"данных в качестве валидных). Соответственно, вместе с проверкой верности ответов необходимо также проверять программу, например valgrind-ом.
В-третьих, даже эффективный алгоритм на определенных тестах может превысить все мыслимые границы быстродействия из-за неоптимального препроцессинга.

Список литературы

[1] Гасфилд Д. *Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология.*/ Пер. с англ. И. В. Романовского. — СПб.: Невский Диалект; БХВ-Петербург, 2003. — 654 с.: ил.