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: А. К. Носов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №4

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

Вариант алгоритма: Поиск одного образца при помощи алгоритма Бойера-Мура.

Вариант алфавита: Числа в диапазоне от 0 до 2 32 - 1.

Формат ввода Искомый образец задаётся 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 и на количество слов или чисел в них, не накладывается.

Формат вывода

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

Для заданий, в которых требуется найти только один образец, следует вывести два числа через запятую: номер строки и номер слова в строке, с которого начинается найденный образец. В заданиях с большим количеством образцов, на каждое вхождение нужно вывести три числа через запятую: номер строки; номер слова в строке, с которого начинается найденный образец; порядковый номер образца.

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 реализовать алгоритм Бойера-Мура для поиска подстроки в строке. Притом алфавит представляет собой множество четырех-байтных целых неотрицательных чисел. Алгоритм Бойера-Мура прикладывает образец к тексту и начинает проверку с конца. Также в алгоритме используется две эвристики(правила) для достижения наибольшей скорости. Первое правило - правило плохого символа. В случае, когда символ текста и образца не совпал, берется буква несовпавшая буква текста и ищется её самое правое вхождение, соответственно образец сдвигается так, чтобы совместить эти буквы. Если буква не присутствует в образце, происходит сдвиг на длина образца. Второе правило - правило хорошего суффикса. Снова, при несовпадении буквы текста и образца, ищется самая правая подстрока, которая равна совпавшему суффиксу, происходит сдвиг так, чтобы совместить эти подстроки. Если же такой подстроки нет, то ищется наибольший префикс, совпадающий с проверенным суффиксом, после сдвига префикс прикладывается на место совпавшего суффикса.

2 Исходный код

Здесь располагается реализация алгоритма Бойера-Мура. Поиск самого правого символа осуществляется с помощью бинарного поиска.

lab4.cpp

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <sstream>

using namespace std;

unordered_map<unsigned long, int> BCTable(const vector<unsigned long>& pat) {
    unordered_map<unsigned long, int> badc;
    for (int i = 0; i < pat.size(); ++i) {
        badc[pat[i]] = i;
    }
    return badc;
}

void computeSuffix(const vector<unsigned long>& pat, vector<int>& suff) {
    int m = pat.size();
    suff.resize(m);
    suff[m - 1] = m;
    int g = m - 1;
    int f = 0;
    for (int i = m - 2; i >= 0; --i) {
        if (i > g && suff[i + m - 1 - f] < i - g) {
            suff[i] = suff[i + m - 1 - f];
        }
        else {
            if (i < g) {
                g = i;
            }
            f = i;
            while (g >= 0 && pat[g] == pat[g + m - 1 - f]) {
                --g;
            }
            suff[i] = f - g;
        }
    }
}

vector<int> GSTable(const vector<unsigned long>& pat) {
    vector<int> gs;
    int m = pat.size();
    vector<int> suff;
    computeSuffix(pat, suff);
```

```
44          gs.resize(m, m);
45          int j = 0;
46          for (int i = m - 1; i >= 0; --i) {
47              if (suff[i] == i + 1) {
48                  for (; j < m - 1 - i; ++j) {
49                      if (gs[j] == m) {
50                          gs[j] = m - 1 - i;
51                      }
52                  }
53              }
54          }
55          for (int i = 0; i <= m - 2; ++i) {
56              gs[m - 1 - suff[i]] = m - 1 - i;
57          }
58          return gs;
59      }
60
61      vector<int> BoyerMoore(const vector<unsigned long>& text, const vector<unsigned
            long>& pat, const unordered_map<unsigned long, int>& badc, const vector<int>&
            gs){
62          int m = pat.size();
63          int n = text.size();
64          vector<int> ocur;
65          int s = 0;
66          while (s <= n - m) {
67              int j = m - 1;
68              while (j >= 0 && pat[j] == text[s + j]) {
69                  --j;
70              }
71              if (j < 0) {
72                  ocur.push_back(s);
73                  s += gs[0];
74              }
75              else {
76                  auto it = badc.find(text[s + j]);
77                  int badc_shift = (it != badc.end()) ? it->second : -1;
78                  int bc_shift = j - badc_shift;
79                  s += max(1, max(bc_shift, gs[j]));
80              }
81          }
82          return ocur;
83      }
84
85      int main() {
86          string line;
87          vector<unsigned long> pat;
88          while (getline(cin, line)) {
89              istringstream iss(line);
90              unsigned long num;
```

```
91              while (iss >> num) {
92                  pat.push_back(num);
93              }
94              if (!pat.empty()) {
95                  break;
96              }
97          }
98
99          if (pat.empty()) {
100             return 0;
101         }
102
103         unordered_map<unsigned long, int> badc = BCTable(pat);
104         vector<int> gs = GSTable(pat);
105
106         int count = 0;
107         while (getline(cin, line)) {
108             count++;
109             istringstream iss(line);
110             vector<unsigned long> text;
111             unsigned long num;
112             while (iss >> num) {
113                 text.push_back(num);
114             }
115             if (text.empty()){
116                 continue;
117             }
118             vector<int> ocur = BoyerMoore(text, pat, badc, gs);
119             for (int s : ocur) {
120                 int i = s + 1;
121                 cout << count << ", " << i << endl;
122             }
123         }
124         return 0;
125     }
```

3 Консоль

```
123 123 123
123 123 123 23424234 324234 2 123 123 123
1,1
1,7
```

4 Тест производительности

Тесты производительности представляют из себя следующее: будет дан шаблон и тексты разной длины. Сравиваться будут время работы наивного алгоритма поиска подстроки в строке и алгоритма Бойера-Мура.

```
BM: 6145 us
Naive: 98323 us
BM: 9625 us
Naive: 129312 us
BM: 16623 us
Naive: 258624 us
```

Как видно из результатов теста, алгоритм Бойер-Мур оказался быстрее наивного на всех тестах. Этот результат был ожидаемым, поскольку сложность алгоритма БМ составляет $\Theta(n)$, где n - длина текста, А сложность наивного алгоритма - $\Theta(n * m)$, потому что для каждой позиции текста длины n мы сравниваем все символы шаблона длины m. Отсюда такая разница в скорости.

5 Выводы

В ходе выполнения лабораторной работы я смог реализовать алгоритм Бойера-Мура нахождения подстроки в строке. Существует много разных алгоритмов поиска подстрок, такое разнообразие обусловлено тем, что, к примеру, алгоритм Бойера-Мура будет долго искать вхождение разных паттернов в текст, когда как алгоритм АхоКорасик способен искать сразу несколько паттернов в тексте.

Список литературы

[1] *Гасфилд Дэн Г 22 Строки, деревья и последовательности в алгоритмах:* Информатика и вычисвычислительная биология / Пер. с англ. И. В. Романовского. — СПб.: Невский Диа-Диалект; БХВ-Петербург, 2003. — 654 с: ил.