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: М. В. Леухин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-20

Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №4

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

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

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

Запрещается реализовывать алгоритмы на алфавитах меньшей размерности, чем указано в задании.

Формат входных данных: 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

В задаче поика подстрок у нас есть две строк: текстовая строка T и шаблонная строка P. Мы хотим найти все вхождения P в T. Поскольку мы хотим найти все вхождения шаблона в текст, решением будут все величины сдвигов P относительно начала T. Другими словами, мы говорим, что шаблон P встречается в тексте со сдвигом s, если подстрока T, которая начинается с t_{s+1} в точности совпадает с шаблоном [1]. Для выполнения задачи используется алгоритм Бойера-Мура, преимущество которого заключается в том, что ценой некоторого количества предварительных вычислений над шаблоном, шаблон сравнивается с исходным текстом не во всех позициях — часть проверок пропускается как заведомо не дающая результата [2]. Засчёт этого достигается сложность $O(n + m)$ в сравнении с наивным алгоритмом за $O(nm)$.

2 Исходный код

Рассмотрим основные 3 идеи, на которых основан алгоритм Бойера-Мура [2]:

1. **Сканирование слева направо, сравнение справа налево.** Совмещается начало текста (строки) и шаблона, проверка начинается с последнего символа шаблона. Если символы совпадают, производится сравнение предпоследнего символа шаблона и т. д. Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и выполняется поиск следующего вхождения подстроки.

2. **Эвристика плохого символа.** Перед началом поиска создаётся таблица, в которой для каждого символа шаблона указвыается последнее его вхождение в шаблон. Предположим теперь, что в процессе поиска символ c текста не совпал с очередным символом шаблона. Очевидно, что в таком случае мы можем сдвинуть шаблон до первого вхождения этого символа c в шаблоне, потому что совпадений других символов точно не может быть. Если в шаблоне такого символа нет, то можно сдвинуть весь шаблон полностью.

3. **Эвристика хорошего суффикса.** Если при сравнении текста и шаблона совпало один или больше символов, шаблон сдвигается в зависимости от того, какой суффикс совпал. Допустим, в процессе поиска совпала подстрока u шаблона. Если существуют такая подстрока, равная u, что она полностью входит в шаблон и идёт слева от несовпавшего символа, то мы можем сразу сдвинуть шаблон к этой подстроке. Если же такой строки не существует, то смещение будет заключаться в выравнивании самого длинного префикса шаблона с подходящей подстрокой текста.

```
1  #include <iostream>
2  #include <vector>
3  #include <unordered_map>
4  #include <algorithm>
5  #include <sstream>
6
7  using namespace std;
8
9  using TText = vector<pair<string, pair<int, int>>>;
10 using TPattern = vector<string>;
11 using TPosition = pair<int, int>;
12
13 int BadSymbolShift(unordered_map<string, int> &table, int i, const string &s) {
14     if (table.count(s) == 0) {
15         return i + 1;
16     }
17     int shift = i - table[s];
```

```
18      return shift > 0 ? shift : 1;
19  }
20
21  vector<TPosition> Find(const TText &text, const TPattern &pattern) {
22      vector<TPosition> answer;
23      if (text.size() < pattern.size() or pattern.size() == 0) {
24          return answer;
25      }
26
27      unordered_map<string, int> stopTable;
28      for (int i = 0; i < pattern.size(); i++) {
29          stopTable[pattern[i]] = i;
30      }
31
32      int i = pattern.size() - 1;
33      while (i < text.size()) {
34          int textIterator = i;
35          int patternIterator = pattern.size() - 1;
36          while (text[textIterator].first == pattern[patternIterator]) {
37              if (patternIterator == 0) {
38                  answer.emplace_back(text[textIterator].second.first, text[textIterator].
                        second.second);
39                  break;
40              }
41              textIterator--;
42              patternIterator--;
43          }
44          i += max(1, BadSymbolShift(stopTable, patternIterator, text[textIterator].first
                ));
45      }
46
47      return answer;
48  }
49
50  int main() {
51
52      std::ios_base::sync_with_stdio(false);
53      std::cin.tie(nullptr);
54      std::cout.tie(nullptr);
55
56      TText text;
57      TPattern pattern;
58
59      string str;
60      getline(cin, str);
61      istringstream patternStream(str);
62
63      while (patternStream >> str) {
64          std::transform(str.begin(), str.end(), str.begin(),
```

```
65                      [](unsigned char c) { return std::tolower(c); });
66          pattern.push_back(str);
67      }
68
69      int line = 0;
70      while (getline(cin, str)) {
71          line++;
72          if (str.empty()) {
73              continue;
74          }
75          istringstream textStream(str);
76          int counter = 1;
77          while (textStream >> str) {
78              std::transform(str.begin(), str.end(), str.begin(),
79                             [](unsigned char c) { return std::tolower(c); });
80              text.emplace_back(str, make_pair(line, counter));
81              counter++;
82          }
83      }
84
85      for (TPosition tp: Find(text, pattern)) {
86          cout << tp.first << ", " << tp.second << "\n";
87      }
88
89  }
```

3 Консоль

```
matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/3da/4lab$ cat 01.t
cat dog cat dog bird
CAT dog CaT Dog Cat DOG bird CAT
dog cat dog bird
matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/3da/4lab$ ./lab4 <01.t
1,3
1,8
```

4 Тест производительности

Тест производительности представляет из себя следующее: сравнение эффективности реализованного алгоритма Бойера-Мура и наивного алгоритма.

```
# pattern: 10 elements
# text: 1000000 elements
matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/3da/4lab$ ./bm <tests/01.t
BM time: 1933297us
matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/3da/4lab$ ./naive <tests/01.t
Naive time: 2055299us
```

Как видно из тестов, алгоритм Бойера-Мура $O(n+m)$ работает действительно быстрее наивного алгоритма $O(nm)$. Однако разница невилика ввиду того, что в данной реализации используется только эвристика плохого символа. К тому же, больший выигрыш алгоритм Бойера-Мура даёт в случаях большого шаблона, так как сдвиги по эвристикам будут больше.

5 Выводы

Выполнив четвёртую лабораторную работу по курсу «Дискретный анализ», я вспомнил уже знакомые мне и узнал новые алгоритмы подиска подстроки в строке, а также научился реализовывать алгоритм Бойера-Мура. Я узнал о том, что из себя представляют эвристики плохого символа и хорошего суффикса и как их использовать. К тому же, при написании программы я больше узнал о работе с потоками ввода в C++.

Список литературы

[1] Томас Х. Кормен *Алгоритмы: вводный курс.* — Издательский дом «Вильямс», 2014. Перевод с английского: И. В. Красиков— 208 с. (ISBN 978-5-8459-1868-0 (рус.)).

[2] *Алгоритм Бойера-Мура — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Алгоритм_Бойера_-_Мура` (дата обращения: 14.05.2022).