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. С. Лагуткина
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №4

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

Вариант алгоритма: Поиск одного образца при помощи алгоритма Кнута-Морриса-Пратта.

Вариант 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 результата: 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. 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 написать реализацию алгоритма Кнута-Мориса-Пратта поиска подстроки в строке. В качестве алфавита выступают слова не более 16 знаков латинского алфавита (регистронезависимые).

Как сказано в [1]: «Хотя этот метод редко используется и часто на практике уступает методу Бойера-Мура (и другим), он может быть просто объяснен, и его линейная оценка времени легко обосновывается. Кроме того, он создает основу для известного алгоритма Ахо-Корасика, который эффективно находит все вхождения в текст любого образца из заданного набора образцов).».

[1].

2 Исходный код

Этапы написания кода:

1. Осуществление ввода
2. Реализация подсчета префикс-функции
3. Реализация алгоритма КМП

Вод текста и шаблона производится по словам. При встрече символа перевода на другую строку, заполняется следующая строка. Таким образом, текст хранится в векторе векторов слов. Далее подсчитывается значение префикс-функции для шаблона. И выполняется поиск по алгоритму КМП: осуществляется проход по всем словам текста и сравнение слов с шаблоном, с учетом значения префикс-функции.

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <cctype>
5
6  std::vector<size_t> CountPrefix(std::vector<std::string>& p) {
7      size_t n = p.size();
8      std::vector<size_t> sp(n);
9      sp[0] = 0;
10     size_t v = 0;
11     for (size_t i = 1; i < n; ++i) {
12         while ((v != 0) && (p[i] != p[v])) {
13             v = sp[v - 1];
14         }
15         if (p[i] == p[v]) {
16             ++v;
17         }
18         sp[i] = v;
19     }
20     sp.push_back(0);
21     return sp;
22 }
23
24 bool EqualPatternWord(std::vector<std::string>& p, std::string& word, size_t i) {
25     if (i >= p.size()) { return false; }
26     return p[i] == word;
27 }
28
29 void KmpSearch(std::vector<std::vector<std::string>>& text, std::vector<std::string>&
       p) {
30     std::vector<size_t> sp = CountPrefix(p);
31     size_t countLine;
```

```
    int countWord;
    size_t j = 0;
    for (size_t line = 0; line < text.size(); ++line) { //
        for (size_t word = 0; word < text[line].size(); ++word) { //
            while ((j > 0) && (!EqualPatternWord(p, text[line][word], j))) {
                j = sp[j - 1];
            }
            if (p[j] == text[line][word]) {
                j++;
            }
            if (j == p.size()) {
                countLine = line;
                countWord = word - p.size() + 1;
                while (countWord < 0) {
                    --countLine;
                    countWord += text[countLine].size();
                }
                std::cout << countLine + 1 << ", " << countWord + 1 << std::endl;
            }
        }
    }
}

bool isSpace(char c) {
    return ((c == ' ') || (c == '\t') || (c == '\n'));
}

enum TState {
    str,
    line
};

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::vector<std::vector<std::string>> text;
    std::vector<std::string> pattern;
    std::vector<std::string> curLine;
    std::string curStr;
    int flagFirstLine = 1;
    TState state = line;
    char letter = getchar();
    while (letter != EOF) {
        switch (state) {
        case line:
            if (!isSpace(letter)) {
                state = str;
                break;
```

```
81              }
82              if (letter == '\n') {
83                  if (flagFirstLine == 1) {
84                      pattern = std::move(curLine);
85                      flagFirstLine = 0;
86                  }
87                  else { text.push_back(std::move(curLine)); }
88                  curLine.clear();
89              }
90              letter = getchar();
91              break;
92
93          case str:
94              if (isSpace(letter)) {
95                  curLine.push_back(std::move(curStr));
96                  curStr.clear();
97                  state = line;
98                  break;
99              }
100             curStr.push_back(std::tolower(letter));
101             letter = getchar();
102             break;
103         }
104     }
105     if (pattern.size() == 0 || text.size() == 0) { return 0; }
106     KmpSearch(text, pattern);
107     return 0;
108 }
```

3 Консоль

```
maria@DESKTOP-6CRUDOR:~$ g++ -pedantic -Wall -std=c++14 -Werror -Wno-sign-compare
-lm da4.cpp -o da4
maria@DESKTOP-6CRUDOR:~$ ./da4
cat dog cat dog bird
CAT dog CaT Dog Cat DOG bird CAT
dog cat dog bird
1,3
1,8
```

4 Тест производительности

Тест производительности представляет из себя следующее: сравнивается время работы алгоритма КМП через префикс-функцию и std::string::find. Тест проводился на сгенеренном тесте в 1000 строк. Время считается в наносекундах.

```
maria@DESKTOP-6CRUDOR:~$ ./da4
1000
kmp search time: 380607500
string find time:1084200
```

Как видно, данная реализация КМП проигрывает std::string::find, так как данная реализация не является наиболее эффективной.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я реализовала алгоритм КМП. Также было необычно в качестве алфавита использовать слова. Примечательно то, что данный алгоритм Кнута-Морисса-Пратта в отличие от алгоритма Ахо-Карасика, может работать в реальном времени.

Список литературы

[1] Д. Гасфилд. *Строки, деревья и последовательности в алгоритмах.* — Издательский дом «Невский диалект», 2003. Перевод с английского: И. В. Романовский. — 6546 с. (ISBN 978-5-94157-321-6 (рус.))