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: Н. П. Ежов
Преподаватель: Н. С. Капралов
Группа: М8О-204Б
Дата:
Оценка:
Подпись:

Москва, 2020

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. 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

Алгоритм Кнута-Морисса-Прутта
Требуется реализовать алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке. Учитывая, что алфавит состоит из регистронезависимых слов не более 16 знаков, нужно уметь правильно представлять переводы строки, пробелы и табуляции.
Согласно [1], алгоритм Кнута-Морриса-Пратта прикладывает образец к тексту и начинает делать сравнение с левого конца. В случае полного совпадения, было найдено вхождение, слвигаем образец на один символ вправо. Если же есть несовпадения, то мы делаем сдвиг по особому правилу, в отличии от алгоритма наивного поиска, который всегда сдвигает на один символ.
Для каждой позиции i определим $sp_i(P)$ как длину наибольшего собственного суффикса $P[1..i]$, который совпадает с префиксом P, причём символы в позициях $i+1$ и $sp_i(P)+1$ различны.
sp - префикс функция. Опишем эффективный алгоритм её построения. Но для начала рассмотрим наивный алгоритм. Наивный алгоритм вычисляет префикс-функцию непосредственно по определению, сравнивая префиксы и суффиксы строк. Обозначим длину строки за n. Будем считать, что префикс-функция хранится в массиве p. Как сказано в [2], такой алгоритм имеет сложность $O(n^2)$. Теперь внесём в алгоритм пару замечаний.
Вносятся несколько важных изменений:
Заметим, что $p[i+1] \leq p[i]+1$. Чтобы показать это, рассмотрим суффикс, оканчивающийся на позиции $i+1$ и имеющий длину $p[i+1]$, удалив из него послдний символ, мы получим суффикс, оканчивающийся на позиции i и имеющий длину $p[i+1]-1$, следовательно неравенство $p[i+1] > 1$ неверно.
Избавимся от явных сравнений строк. Пусть мы вычислили $p[i]$, тогда, если $s[i+1] = s[p[i]]$, то $p[i+1] = p[i]+1$. Если окажется, что $s[i+1] \neq s[p[i]]$, то нужно попытаться попробовать подстроку меньшей длины. Хотелось бы сразу перейти к такому бордеру наибольшей длины, для этого подберем такое k, что $k = p[i]-1$. Делаем это следующим образом. За исходное k необходимо взять $p[i-1]$, что следует из первого пункта. В случае, когда символы $s[k]$ и $s[i]$ не совпадают, $p[k-1]$ - следующее потенциальное наибольшее значение k. Последнее утверждение верно, пока $k > 0$, что позволит всегда найти его следующее значение. Если $k = 0$, то $p[i] = 1$ при $s[i] = s[1]$, иначе $p[i] = 0$.

2 Исходный код

В файле kmp.hpp содержутся функции для КМП, а также структура TWord, которая используется для хранения слова, его номера в строке, и номер строки, в которой он находится.

Листинг kmp.hpp

```
1  //
2  // kmp.hpp
3  //
4
5  #ifndef LAB4_KMP_HPP
6  #define LAB4_KMP_HPP
7  #include <vector>
8  #include <string>
9  namespace NSearch {
10     struct TWord{
11         std::string Word;
12         long long WordId, StringId;
13     };
14     std::string tolower(std::string &data){
15         int size = data.size();
16         for (int i = 0; i < size; ++i){
17             data[i] = std::tolower(data[i]);
18         }
19         return data;
20     }
21     std::vector<long long> PrefixFunction(const std::vector<TWord> &input) {
22         std::vector<long long> res(input.size());
23         long long length = input.size();
24         for (long long i = 1; i < length; ++i) {
25             long long j = res[i - 1];
26             while(j > 0 && input[i].Word != input[j].Word){
27                j = res[j - 1];
28             }
29             if (input[i].Word == input[j].Word){
30                 ++j;
31             }
32             res[i] = j;
33         }
34         return res;
35     }
36     std::vector<long long> KMP(std::vector<TWord>&pattern, std::vector<TWord> &text,
           std::vector<long long> &prefix){
37         long long patternSize = pattern.size();
38         long long textSize = text.size();
39         long long i = 0;
```

```
        std::vector<long long> answer;
        if (patternSize > textSize || patternSize == 0) {
            return answer;
        }
        while (i < textSize - patternSize + 1) {
            long long j = 0;
            while (j < patternSize && pattern[j].Word == text[i + j].Word) {
                ++j;
            }
            if (j == patternSize) {
                answer.push_back(i);
            }
            else{
                if (j > 0 && j > prefix[j - 1]) {
                    i = i + j - prefix[j - 1];
                    continue;
                }
            }
            ++i;
        }
        return answer;
    }
}
#endif //LAB4_KMP_HPP
```

В файле *main.cpp* будем посимвольно считывать текст. После считывания запускаем функцию поиска КМП, после чего выводим полученные ответы.

Листинг main.cpp

```
//
// main.cpp
//
#include <iostream>
#include "kmp.hpp"
int main(){
    bool flagPattern = true;
    std::cin.tie(nullptr);
    std::cout.tie(nullptr);
    std::ios_base::sync_with_stdio(false);
    std::vector<NSearch::TWord> pattern;
    std::vector<NSearch::TWord> text;
    NSearch::TWord input;
    std::string word;
    char c = getchar();
    while (c > 0) {
        if (c == '\n') {
            if (!input.Word.empty()) {
                text.push_back(input);
```

```
20                  input.Word.clear();
21              }
22              ++input.StringId;
23              if (flagPattern) {
24                  std::swap(pattern, text);
25                  flagPattern = false;
26                  input.StringId = 1;
27              }
28              input.WordId = 1;
29          } else if (c == '\t' || c == ' ') {
30              if (!input.Word.empty()) {
31                  text.push_back(input);
32                  input.Word.clear();
33                  ++input.WordId;
34              }
35          } else {
36              if ('A' <= c and c <= 'Z') {
37                  c = c + 'a' - 'A';
38              }
39              input.Word += c;
40          }
41          c = getchar();
42      }
43      if (!input.Word.empty()) {
44          text.push_back(input);
45      }
46      std::vector<long long> prefix = NSearch::PrefixFunction(pattern);
47      std::vector<long long> ans = KMP(pattern, text, prefix);
48      for (long long & id : ans) {
49          std::cout << text[id].StringId << ", " << text[id].WordId << '\n';
50      }
51      return 0;
52  }
```

3 Консоль

```
nezhov@LAPTOP-VDTFKP8R:/mnt/c/Users/nikit/Diskran/lab4$ cat tests/01.t
cat dog cat dog bird
CAT dog CaT Dog Cat DOG bird CAT
dog cat dog bird
nezhov@LAPTOP-VDTFKP8R:/mnt/c/Users/nikit/Diskran/lab4$ cat tests/01.a
1,3
1,8
nezhov@LAPTOP-VDTFKP8R:/mnt/c/Users/nikit/Diskran/lab4$ make
g++ -std=c++17 -O3 -Wextra -Wall -Werror -Wno-sign-compare -Wno-unused-result
-pedantic -o solution main.cpp
nezhov@LAPTOP-VDTFKP8R:/mnt/c/Users/nikit/Diskran/lab4$ ./solution <tests/01.t
>res.txt
nezhov@LAPTOP-VDTFKP8R:/mnt/c/Users/nikit/Diskran/lab4$ diff res.txt tests/01.a
```

4 Тест производительности

Реализованный алгоритм Кнута-Морриса-Пратта с использнованием прфикс функции сравнивается с наивным алгоритмом поиска. Замеры производятся на тестах из 10^3 слов, 10^4 или 10^5. Длина образца в первом тесте - 10, во втором - 25, в третьем - 100

```
nezhov@LAPTOP-VDTFKP8R:/mnt/c/Users/nikit/Diskran/lab2$ ./benchmark <tests/03.t
==============START============
Naive: 0.822 ms
KMP: 0.049 ms
==============END==============
nezhov@LAPTOP-VDTFKP8R:/mnt/c/Users/nikit/Diskran/lab2$ ./benchmark <tests/04.t
==============START============
Naive: 5.418 ms
KMP: 0.564 ms
==============END==============
nezhov@LAPTOP-VDTFKP8R:/mnt/c/Users/nikit/Diskran/lab2$ ./benchmark <tests/05.t
==============START============
Naive: 25.715 ms
KMP: 2.412 ms
==============END==============
```

Видно, что наивный алгоритм поиска почти на порядок проигрывает алгоритму Кнута-Морриса-Пратта. Классический алгоритм допускает лишние сравнения на этапе поиска образца в тексте, а алгоритм с применение префикс функции - нет. Обработка таких сравнений длиться дольше, чем предпроцессинг префикс функции.

5 Выводы

Во время выполнения лабораторной работы я изучил алгоритм Кнута-Морриса-Пратта с предпроцессингом через префикс функцию.
Задачи поиска подстроки часто встречаются в жизни, один из очевидных примеров это поиск контента в Интернете по ключевому слову или по ключевой фразе.
Наивный алгоритм поиска нельзя использовать для поиска, т.к. он действует слишком медленно. С другой стороны, КМП не может эффективно находить несколько образцов в тексте, но с этим хорошо справляется алгоритм Ахо-Корасика.

Список литературы

[1] Гасфилд Дэн. *Строки, деревья и последовательности в алгоритмах: Информатика и вычислетльная биология.* — Издательский дом «БХВ-Петербург». Перевод с английского: И.В. Романовский — 654 с.(дата обращения: 03.12.2020).

[2] *NEERC ITMO.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Префикс-функция` (дата обращения: 03.12.2020).