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: Ф. А. Иванов
Преподаватель: И. Н. Симахин
Группа: М8О-208Б-19

Дата: 20.10.2021
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. 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 написать реализацию алгоритма КМП поиск подстроки в строке.

Прямой поиск vs КМП
При прямом поиске происходит посимвольное сравнение, и в случае несовпадения - сдвиг образца на одне позицию по строке и откат индекса по строке на начальную позицию сравнения +1.
Алгоритм КМП позволяет сдвинуть образ более чем на 1 символ по строке, а индекс по строке никогда не откатывается

Этапы работы алгоритма КМП

1. Формирование массива π, используемого при сдвиге вдоль строки.
2. Поиск образа в строке.

Префикс-функция
Префикс-функция для i-ого символа образца возвращает значение, равное максимальной длине совпадающих префикса и суффикса подстроки в образе, которая заканчивается i-ым символом. Это значение будем хранить в $\pi[i]$.

Программная реализация этапа 1
индексы: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8
образец: a | b | b | a | a | b | b| a | b
π: 0 | 0 | 0 | 1 | 1 | 2 | 3 | 4 | 2

если $a_i \neq a_j$, тогда, если $j = 0$, то $\pi[i] = 0; i{+}{+}$;
иначе (т.е.$j \neq 0) j = \pi[j-1]$;
иначе (т.е. $a_i = a_j) \pi[i] = j + 1; i{+}{+}; j{+}{+}$;

Программная реализация этапа 2
если $t_k = a_l$, тогда $\{k{+}{+}; l{+}{+}$; если l=n, то образец найден$\}$
иначе (т.е. $t_k \neq a_l$), если $l = 0$, тогда $\{$ $k{+}{+}$; если $k = m$, то образа в строке нет$\}$
иначе (т.е. $l \neq 0$) $l = \pi[l-1]$;

2 Исходный код

```
#include <iostream>
#include <vector>

int main(){

    std::vector<std::pair<long, long>> answer; // array for storage answers
    std::vector<std::string> pattern; // substring
    std::vector<std::string> text; // all text divaded on words
    std::vector<std::pair<long, long>> mapText; // map of text for quick finding of
        positions of words

    // substring reading unit
    std::string word;
    char c;
    c = getchar();
    while(c != '\n'){
        if(c == ' ' || c == '\t'){
            if(!word.empty()){
                pattern.push_back(word);
                word.clear();
            }
        }else {
            word += std::tolower(c);
        }
        c = getchar();
    }
    if(!word.empty()){
        pattern.push_back(word);
        word.clear();
    }

    // text reading unit and filling of text map
    long numStr = 1, posWord = 0;
    c = getchar();
    while(c != EOF){
        if(c == ' ' || c == '\t'){
            if(!word.empty()){
                text.push_back(word);
                posWord++;
                mapText.push_back(std::make_pair(numStr, posWord));
                word.clear();
            }

        } else if(c == '\n'){
            if(!word.empty()){
                text.push_back(word);
                posWord++;
```

```
47                mapText.push_back(std::make_pair(numStr, posWord));
48                word.clear();
49            }
50            numStr++;
51            posWord = 0;
52        } else {
53            word += std::tolower(c);
54        }
55        c = getchar();
56    }
57    if(!word.empty()){
58        text.push_back(word);
59        posWord++;
60        mapText.push_back(std::make_pair(numStr, posWord));
61        word.clear();
62    }
63
64    //algorithm KMP
65    // first part: template creating
66    //0 1 2 3 4 5 6 7 8
67    //a b b a a b b a b
68    //0 0 0 1 1 2 3 4 2
69    long sizePattern = pattern.size();
70    long arr[sizePattern];
71    arr[0] = 0;
72    long j = 0;
73    long i = 1;
74    while(i < sizePattern) {
75        if(pattern[i] == pattern[j]) { // a_i == a_j
76            arr[i] = j+1; i++; j++;
77        } else if(j == 0) { // a_i != a_j & j == 0
78            arr[i] = 0; i++;
79        } else {
80            j = arr[j-1];
81        }
82    }
83
84    //second part
85    long sizeText = text.size();
86    long k = 0;
87    long l = 0;
88    i = 0;
89    while(k < sizeText) {
90        if(text[k] == pattern[l]) {
91            k++; l++;
92            if(l == sizePattern){ // end: -> patern is finded
93                answer.push_back(mapText[k-l]);
94                l = arr[l-1];
95            }
```

```
96          } else if(l == 0) {
97              k++;
98          } else {
99              l = arr[l-1];
100         }
101     }
102
103     //print all answers
104     for(long i = 0; i < answer.size(); ++i){
105         std::cout << answer[i].first << ", " << answer[i].second << "\n";
106     }
107
108     return 0;
109 }
```

3 Консоль

```
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$ g++ main.cpp -o main
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$ cat 00:test_01.txt
cat dog cat dog bird
CAT dog CaT Dog Cat DOG bird CAT
dog cat dog bird
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$ ./main <00:test_01.txt
1,3
1,8
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$
```

4 Тест производительности

В ходе выполнения теста производительности хочу показать, что сложность алгоритма КМП действительно $O(n)$.
Имеется 7 подготовленных тестов, полученных с помошью генератора тестов. 1 тест:

- подстрока 16 слов;
- текст 10 слов.

2 тест:

- подстрока 16 слов;
- текст 100 слов.

3 тест:

- подстрока 16 слов;
- текст 1000 слов.

4 тест:

- подстрока 16 слов;
- текст 10 000 слов.

5 тест:

- подстрока 16 слов;
- текст 100 000 слов.

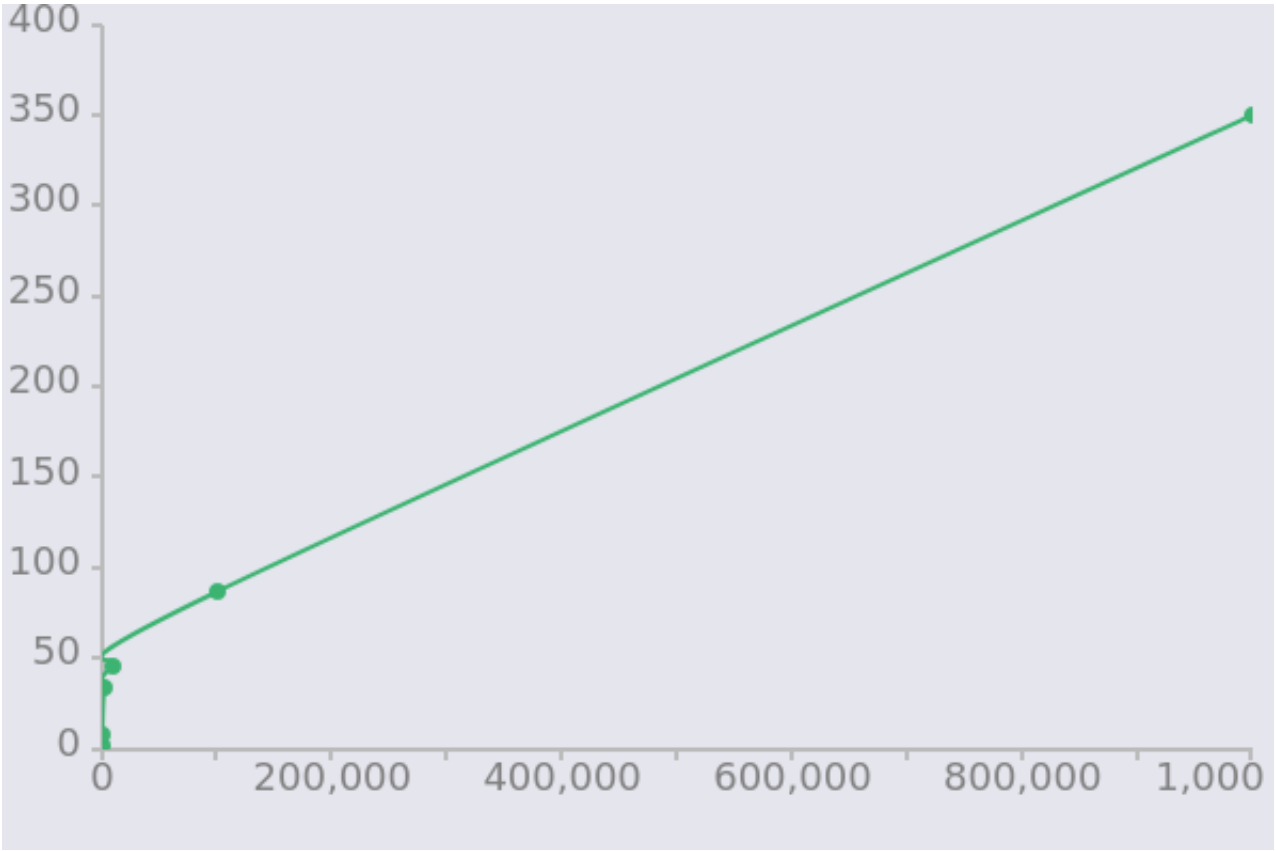
6 тест:

- подстрока 16 слов;
- текст 1 000 000 слов.

7тест:

- подстрока 16 слов;
- текст 10 000 000 слов.

```
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$ g++ main.cpp -o main
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$ time ./main <test_01.txt

real    0m0,002s
user    0m0,003s
sys     0m0,000s
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$ time ./main <test_02.txt

real    0m0,008s
user    0m0,008s
sys     0m0,000s
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$ time ./main <test_03.txt

real    0m0,034s
user    0m0,007s
sys     0m0,000s
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$ time ./main <test_04.txt

real    0m0,046s
user    0m0,014s
sys     0m0,005s
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$ time ./main <test_05.txt

real    0m0,087s
user    0m0,051s
sys     0m0,009s
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$ time ./main <test_06.txt

real    0m0,351s
user    0m0,257s
sys     0m0,068s
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$ time ./main <test_07.txt

real    0m3,616s
user    0m2,962s
sys     0m0,451s
sage@sage-HP-ProBook-440-G5:~/Desktop/DA/3_sem/DA_LAB_04$
```


400
350
300
250
200
150
100
50
0
0
200,000
400,000
600,000
800,000
1,000

5 Выводы

Выполнив четвертую лабораторную работу по курсу «Дискретный анализ», я узнал, что существует множество алгоритмов поиска подстроки в тексте. Применяется в виде встроенной функции в текстовых редакторах, СУБД, поисковых машинах, языках программирования и т.п..
От меня требовалось реализовать алгоритм КМП. Я смог написать оптимальную реализацию этого алгоритма. На графике было показанно, что на больших тестах >1 000 слов алгоритм начинает наботать за линейное время $O(m + n)$.

Наиболее распространенные алгоритмы решения этой задачи: КМП и Ахо-Корасик.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))