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: Д.В. Семенов
Преподаватель: А.А. Кухтичев
Группа: М8О-206Б
Дата:
Оценка:
Подпись:

Москва, 2019

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

1 Описание

Нужно составить программу, которая находит в тексте заданный образец с помощью алгоритма Бойера-Мура.

Формат входных данных: Искомый образец задается на первой строке входного файла. Затем следует текст, состоящий из слов, в котором нужно найти заданный образец. Никаких ограничений на длину строк, равно как на количество слов в них, ненакладывается.

Формат выходных данных: В выходной файл нужно вывести информацию о всех вхождениях искомого образца в обрабатываемый текст: по одному вхождению на строчку. 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.

Алгоритм Бойера Мура основан на двух правилах: поиск шаблона ведется слева направо, а сравнивание шаблона с подстрокой идет справа налево. В случае несовпадения (или, наоборот, полного попадания) используются две заранее вычисляемые функции: функция плохого символа и функция хорошего суффикса.

2 Исходный код

Сначала мы считываем искомую строку и вычисляем с помощью нее все необходимые функции (плохого символа $map < char, int > bad\_char$ и хорошего суффикса $vector < int > good\_suffix$). Выполняется это все в конструкторе класса для более удобной работы. Особое внимание следует уделить правильному считыванию литер. Для организации хранения позиций слов я использовал очередь, состоящую из пар <номер строки, номер слова>.

Затем используем простой алгоритм обработки данных:

1. Будем считывать и подготавливать текст, пока его размер не будет равен шаблону
2. Сравним шаблон и подстроку
3. Сдвинем на нужное количество позиций и удалим ненужные символы
4. Снова считаем текст и подготовим его

```
1  void work2(){
2
3        read_text();
4
5
6        do {
7            prepare_text();
8
9            if (textsize < pattsize) {
10               read_text();
11               //prepare_text();
12           }
13           //make_text();
14
15           compare();
16           moving();
17
18           prepare_text(); // CHANGED
19
20           read_text();
21
22
23       } while (textsize == pattsize);
24
25    }
26
27  void compare(){
28        int i;
```

```
29
30          for (i = pattsize-1; i >=0 && pattern.pattern[i] == text[i]; i--);
31
32          if (i < 0) {
33              pair<int, int> mytmp = pos.front();
34
35              cout << mytmp.first << ", " << mytmp.second << endl;
36              move = pattern.good_suffix[0];
37
38          } else {
39              map<char,int>::iterator it;
40              it = pattern.bad_char.find(text[i]);
41              if (it == pattern.bad_char.end()) {
42                  move = pattsize-1;
43              } else {
44                  move = it->second;
45              }
46
47              if (pattern.good_suffix[i+1] > (int)(move)) {
48                  move = pattern.good_suffix[i];
49              }
50          }
51      }
52
53  void prepare_text() {
54
55          if (text[0] == ' '){
56              if (char_before_all != ' ') {
57                  pos.pop();
58              }
59              text.erase(0, 1);
60              textsize--;
61          }
62      }
```

3 Консоль

```
denis@denis-Extensa-2510G:~/labs_3sem/labs_da/4lab_debug$ ./4lab <test5.txt

1,16

3,12

11,3

13,26

denis@denis-Extensa-2510G:~/labs_3sem/labs_da/4lab_debug$ ./4lab <test3.txt

1,2

4,1

4,3
```

Пример теста 1:
he llo he
he
llo
he llo he llo he
Полученный ответ:
1, 1
3, 1
3, 3
Пример теста 2:
a a b a a c a a b a a b a
a a b a a b
a
a c a a b a a b a a b a a c
a a b a a b a a b
a a b a a b
Полученный ответ:

1, 4
4, 9

4 Тест производительности

```
1 clock_t beg = clock();
2  /* some work */
3 clock_t end = clock();
4 std::cout << "time is " << end - beg << endl;
```

Размер текста	Размер шаблона	нативный	BM
796	4	451	1194
12225	18	8760	3894
38805	101	62989	10548

Как мы видим, на больших объемах текста и больших шаблонах наш алоритм работает в разы быстрее.

5 Выводы

Выполнив четвертую лабораторную работу по курсу «Дискретный анализ», я изучил основные алгоритмы поиска подстроки в строке. Алгорит Бойера-Мура считается наиболее быстрым из всех, он очень быстр на "хороших"данных. Разве что на коротких текстах выигрыш не оправдает предварительных вычислений. Также, у этого алгоритма есть множество улучшений, таких как Турбо БМ, Бойера-Мура-Хорспула и др.

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.))

[2] *Алголист http://algolist.manual.ru/search/esearch/bm.php*