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: В. П. Будникова
Преподаватель: А. А. Кухтичев
Группа: М8О-209Б-19

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 Бойера-Мура

Вариант алфавита: `Числа в диапазоне от` 0 `до` $2^{32} - 1$

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. 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

Как сказано в [1]: «Алгоритм поиска строки Бойера — Мура — алгоритм общего назначения, предназначенный для поиска подстроки в строке. Преимущество этого алгоритма в том, что ценой некоторого количества предварительных вычислений над шаблоном (но не над строкой, в которой ведётся поиск), шаблон сравнивается с исходным текстом не во всех позициях — часть проверок пропускаются как заведомо не дающие результата.».
В алгоритме Бойера-Мура происходит предварительная обработка паттерна и последующее его сравнение с текстом. Сравнение происходит справа-налево, а паттерн сдвигается слева-направо. Когда происходит сравнение паттерна с текстом и находим не совпадающую позицию, выбирается сдвиг текста, который будет максимальным числом из 1, сдвига, который мы определим с помощью плохого символа и сдвига, который мы определим с помощью правила хорошего суффикса. Перед началом сравнения необходимо создать две структуры данных, которые будут хранить данные для этих правил. Для правила плохого символа создадим std::map, где ключ - элемент паттерна, значение - вектор его позиций в паттерне. Пройдемся по паттерну и заполним std::map. Для правила хорошего суффикса необходимо сначала сделать реверс паттерна, чтобы считать z-функцию от конца(искать совпадающие суффиксы). Далее мы реверсируем полученный вектор. Создадим еще один вектор, чтобы для каждой позиции знать ее сдвиг по правилу хорошего суффикса. Заполнение этого вектора происходит так: Мы проходимся по вектору z-функции и мы находим не нулевое значение(значит суффикс такой длины совпадает с этой позиции), следовательно, записываем в позицию, с которой идет совпадение позицию элемента вектора z-функции, в которой хранится это не нулевое значение. Это означает, что сравнение теперь будет продолжаться с этой позиции паттерна. При считывании данных мы записываем из в структуру TNum, где хранятся строчка, на которой стоит элемент, позиция элемента в строчке и значение элемента. Также для удобстра сравнения переопределим оператор != для этой структуры и элемента паттерна.

2 Исходный код

Структуры	
TNum	Структура для хранения элемента текста
Функции	
size_t ToInt(size_t a)	Функция для преобразования в число
size_t BadSymbRule(const std::vector<size_t> &in, const size_t &j)	Функция Правила плохого символа
void GoogSuffRule(const std::vector<size_t> &nz, const std::vector<size_t> &pattern, std::vector<size_t> &l)	Функция Правила Хорошего суффикса
void BM(const std::vector<size_t> &pattern, const std::vector<TNum> &text, const std::map<size_t, std::vector<size_t» &forBadSuff, const std::vector<size_t> &l)	Функция алгоритмя Бойера-Мура

```
int main() {
    char a;
    size_t num = 0;
    std::vector<size_t> pattern;
    std::vector<TNum> text;
    bool fl = true;
    int count = 0;
    while ((a = getchar()) != '\n') {
        if (a == EOF) {
            return 0;
        }
        if (a >= '0' && a <= '9') {
            num = num * 10 + ToInt(a);
            fl = true;
            count++;
        } else if (a == ' ' || a == '\t') {
            fl = false;
        }
        if (!fl && count != 0) {
            pattern.push_back(num);
            num = 0;
            count = 0;
            fl = true;
        }
    }
    if (count != 0) {
        pattern.push_back(num);
```

```
28          num = 0;
29      }
30      size_t str = 1;
31      size_t pos = 1;
32      fl = true;
33      while ((a = getchar()) != EOF) {
34          if (a == '\n') {
35              if (!fl) {
36                  text.push_back(TNum{num, str, pos});
37                  num = 0;
38                  fl = true;
39              }
40              ++str;
41              pos = 1;
42          } else if (a >= '0' && a <= '9') {
43              num = num * 10 + ToInt(a);
44              fl = false;
45          } else if (!fl) {
46              text.push_back(TNum{num, str, pos});
47              fl = true;
48              ++pos;
49              num = 0;
50          }
51      }
52      if (count != 0 && !fl) {
53          text.push_back(TNum{num, str, pos});
54      }
55      if (pattern.size() == 0 || text.size() == 0 || pattern.size() > text.size()) {
56          return 0;
57      }
58      std::map<size_t, std::vector<size_t>> forBadSuff;
59      for(size_t i = 0; i < pattern.size(); ++i) {
60          if (forBadSuff.find(pattern[i]) == forBadSuff.end()) {
61              forBadSuff.insert(std::pair<size_t, std::vector<size_t>>{pattern[i], {i}});
62          } else {
63              forBadSuff.find(pattern[i])->second.push_back(i);
64          }
65      }
66      std::reverse(pattern.begin(), pattern.end());
67      std::vector<size_t> nz;
68      size_t l = 0;
69      size_t r = 0;
70      for (size_t i = 0; i < pattern.size() ; ++i) {
71          nz.push_back(0);
72      }
73      for (size_t i = 1; i < pattern.size(); ++i) {
74          if (r > i) {
75              if (r - i < nz[i - l] && r - i > 0) {
76                  nz[i] = r - i;
```

```
77              } else if (r - i > nz[i - l] && nz[i - l] > 0) {
78                  nz[i] = nz[i - l];
79              }
80          } else {
81              nz[i] = 0;
82          }
83          while (i + nz[i] < pattern.size() && pattern[nz[i]] == pattern[i + nz[i]]) {
84              ++nz[i];
85          }
86          if (i + nz[i] > r) {
87              l = i;
88              r = i + nz[i];
89          }
90      }
91      std::reverse(pattern.begin(), pattern.end());
92      std::reverse(nz.begin(), nz.end());
93      std::vector<size_t> lbs;
94      for (size_t i = 0; i < pattern.size() ; ++i) {
95          lbs.push_back(0);
96      }
97      GoogSuffRule(nz, pattern, lbs);
98      BM(pattern, text, forBadSuff, lbs);
99      pattern.clear();
100     text.clear();
101     forBadSuff.clear();
102     lbs.clear();
103     nz.clear();
104 }
```

3 Консоль

```
Lera:B-M valeriabudnikova$ make
g++ -std=c++11 -pedantic -Wall -std=c++11 -Werror -Wno-sign-compare -O2 -lm
main.cpp -o bm
Lera:B-M valeriabudnikova$ cat 1.txt
11 45 11 45 90
0011 45 011 0045 11 45 90    11
45 11 45 90          44806 26913 16038




     9173        56484 31987 80260 14629 56606 19030 26807
42070 71754   000      00346 46059 28116       39197 09291
Lera:B-M valeriabudnikova$ ./bm <1.txt
1,3
1,8
Lera:B-M valeriabudnikova$
```

4 Тест производительности

Сравним время работы алгоритма поиска подстроки Бойера-Мура моей реализации и метода find из std::string.
Так как find находит позицию первого вхождения подстроки, то сначала ограничим мой алгоритм до поиска первого вхождения. В файле > 300000 на каждой от 4 до 20 элементов. Первый встречающийся паттерн находится на 270000 строчке.

```
Lera:B-M valeriabudnikova$ make run
./bm <1.txt
Time B-M: 1 ms
Lera:B-M valeriabudnikova$ make run1
./test <1.txt
Time std::string -find: 2 ms
Lera:B-M valeriabudnikova$ make run1
./test <1.txt
Time std::string -find: 1 ms
```

Как можно увидеть время затраченное на поиск одинаково(учитывая погрешность). По алгоритму Бойера-Мура до совпадения каждый раз будем сдвигаться на длину паттерна.

Теперь проверим работу алгоритма на файле, количество строк в котором 3000000, причем паттерн равномерно встречсается после 270000 строчки. Также после первого нахождения вхождения find будет искать паттерн со следующего элемента текста. Сравним время вместе с выдачей, чтобы убедиться, что мы нашли одитнаковое колво элементов.

Lera:B-M valeriabudnikova$ make run ./bm < 1.txt	Lera:B-M valeriabudnikova$ make run1 ./test < 1.txt
273952, 1	27534196
548578, 4	55135500
548580, 14	55135774
823206, 4	82737078
823208, 14	82737352
1097834, 4	110338656
1097836, 14	110338930
1372462, 4	137940234
1372464, 14	137940508
1647090, 4	165541812
1647092, 14	165542086

1921718, 4	193143390
1921720, 14	193143664
2196346, 4	220744968
2196348, 14	220745242
2470974, 4	248346546
2470976, 14	248346820
2745602, 4	275948124
2746279, 4	276015456
Time B-M: 0.378964 s	Time std::string - find: 0.699916 s

Мой алгоритм почти в 2 раза выиграл find.

5 Выводы

Выполнив четвертую лабораторную работу по курсу «Дискретный анализ», я научилась я научилась реализовывать алгоритм Бойера-Мура для поиска подстроки в строке. Также я попрактиковалась в использовании контейнеров stl. Реализация данного алгоритма далась мне гораздо проще, чем реализация B-Дерева в прошлой лабораторной работе. Самым сложным для меня в данной лабораторной работе оказалось правильное считывание символов, так как мне нужно было считать символ и преобразовать его в число, при этом не учитывать передние нули, пробелы, табуляции и переводы строк, при этом необходимо было следить за подсчетом строк и позиций элемента.

Список литературы

[1] *Алгоритм Бойера — Мура— Википедия.*
URL: `https://ru.wikipedia.org/wiki/Алгоритм_Бойера_-_Мура` (дата обращения: 21.12.2020).