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: Е. С. Пищик
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

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.

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 реализовать алгоритм Бойера-Мура, основная идея алгоритма состоит в том, что pattern прикладывают к text слева-направо, а сами символы сравнивают справо-налево, при совпадении продолжаем сравнивать до конца, если произошло несовпадение символов, то мы делаем сдвиг на длину = максимуму из сильного правила хорошего суффикса, сильного правила плохого символа и 1. Для подсчета этих сдигов потребуется посчитать функции - Z, N, l, L и массив, хранящий самое правое вхождение символа паттерна в текст. При данной реализации сложность алгоритма Бойера-Мура = O(n).

2 Исходный код

main.cpp	
int main()	Основная функция, собирающая все части программы в единое целое.
bm.cpp	
std::vector<size_t> ZFunc(std::vector<unsigned int>&)	Функция, возвращающая вектор, хранящий значения Z-функции для pattern.
std::vector<size_t> NFunc(std::vector<unsigned int>&)	Функция, возвращающая вектор, хранящий значения N-функции для pattern(зеркальная Z-функция).
std::pair<std::vector<size_t>, std::vector<size_t>> LFunctions(std::vector<unsigned int>&)	Функция, возвращающая пару векторов, первый хранит предпосчитанную L-функцию, второй l-функцию для pattern.
void BM(std::vector<std::pair<std::pair <size_t, size_t>, unsigned int>> const& text, std::vector<unsigned int>& pattern)	Функция, выполняющая алгоритм Боуера-Мура и выводящая результат на экран.
pattern_parse.cpp	
void PatternParse(std::vector<unsigned int>&)	Функция, считывает первую строку и разбивает ее на vector чисел.
text_parse.cpp	
void TextParse(std::vector<std::pair <std::pair<size_t,size_t>, unsigned int>>&)	Функция, считывающая оставшиеся строки и разбивает ее на vector пар - пара(номер строки, номер слова в строке), символ text.

3 Консоль

```
pe4eniks$ cat test.txt
11 45 11 45 90
0011 45 011 0045 11 45 90    11
45 11 45 90
pe4eniks$ ./solution < test.txt
1,3
1,8
```

4 Тест производительности

Тест состоит из двух частей:

1. Дан pattern из символа 50 и text 10^5 строк по 50 символов значением 50. Сравним собственный алгоритм БМ и написанный алгоритм поиска подстроки в строке C++.

2. Дан pattern из 3 случайных символов от 0 до 2 и text 10^5 строк по 50 символов от 0 до 2. Сравним собственный алгоритм БМ и написанный алгоритм поиска подстроки в строке C++.

```
pe4eniks$ ./solution <banchmark_bad.txt
C++ find: 4.74135 seconds
My BM: 5.11373 seconds

pe4eniks$ ./solution <banchmark_normal.txt
C++ find: 4.12991 seconds
My BM: 3.93231 seconds
```

Как можно увидеть, время работы примерно равно +- погрешность, в различных комбинациях text и pattern время работы будет меняться, но в общем случае БМ быстрее, ибо тут было очень много повторяющихся комбинаций -> небольшие сдвиги. Даже тут видно, что когда каждый раз двигаться на 1 - БМ работает хуже, но вот когда уже идут комбинации из 3 символов pattern в text - БМ показывает себя лучше.

5 Выводы

Выполнив четвёртую лабораторную работу по курсу «Дискретный анализ», я познакомился различными алгоритмами поиска подстроки в строке: КМП, БМ, Ахо-Карасик, Апостолико-Джанкарло, узнал преимущества и недостатки каждого. Очень важно уметь быстро искать подстроку в строке, т.к. это достаточно часто может пригодиться в жизни. Реализация БМ при помощи СППС и СПХС дают сложность O(n), что достаточно быстро. Также получше познакомился с stl и её различными контейнерами.

Список литературы

[1] Z-функция

[2] БМ wiki