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: И. А. Мариничев
Преподаватель: Н. С. Капралов
Группа: М8О-208Б

Дата: 28.12.20
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: 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.

Затем следует текст, состоящий из слов, 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.

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

Согласно [2], алгоритм Бойера-Мура последовательно прикладывает образец P к тексту T и проверяет совпадение символов P с прилежащими символами T. Когда проверка завершается, P сдвигается вправо по T. Но алгоритм Бойера-Мура в отличие от других подобных алгоритмов использует три идеи: **просмотр справа налево**, **правило сдвига по плохому символу** и **правило сдвига по хорошему суффиксу**. Совместный эффект этих идей позволяет алгоритму проверять меньше, чем n + m символов и после некоторых улучшений работать за линейное время в худшем случае (сублинейность). Итак, рассмотрим используемые эвристики:

1. **Правило плохого символа:** Определим крайнее справа вхождение каждого символа образца, и в случае несовпадения $pattern[i]$ с $text[i+j]$ сдвинем шаблон вправо так, чтобы ближайший слева от «места несовпадения» символ $text[i+j]$ в образце встал на место символа $pattern[i]$.

2. **Правило хорошего суффикса:** При несовпадении в позиции i, сдвинем шаблон вправо так, чтобы его суффикс $pattern[i+1 \ldots m]$ совпал с точно такой же подстрокой, ближайшей слева от «места несовпадения», притом слева этой подстроки обязательно стоит символ, не равный $pattern[i]$.

С помощью данных правил подсчитаем возможные сдвиги и выберем из них максимальный.

2 Исходный код

Проект состоит из 2 файлов:

- **main.cpp**: главный файл в котором происходит считывание данных и сам поиск;

- **preprocessing.hpp**: реализация правила плохого символа и правила хорошего суффикса;

Таблица методов и функций

main.cpp	
Функция	Значение
void Lowercase()	Функция, обеспечивающая регистронезависимость алфавита.
void ReadInput()	Функция, в которой происходит считывание образца и текста, а также разделение текста на слова с вычислением позиций слов в тексте.
int main()	Главная функция, в которой происходит поиск образца в тексте, и вывод найденных позиций.
preprocessing.hpp	
Функция	Значение
void BadCharRule()	Правило плохого символа.
void GoodSuffixRule()	Правило хорошего суффикса.

3 Тест производительности

Я решил сравнить свою реализацию алгоритма Бойера-Мура с наивным алгоритмом поиска. Тестирование происходило на тестах с примерно миллионом слов. В первом тесте текст из часто повторяющихся слов, во втором - из редко повторяющихся. Время выводится микросекундах. Для измерения времени использовалась бибилиотека **chrono**.

```
// Тест 1. Текст,состоящий из 10^6 слов,часто повторяющихся.
ivan@Laptop-IM:/mnt/c/Users/Иван/projects/da_labs/da_lab4/benchmark$ ./benchmark_BM
<../tests/08.t
Boyer Moore Algorithm ms=783563
ivan@Laptop-IM:/mnt/c/Users/Иван/projects/da_labs/da_lab4/benchmark$ ./benchmark_BM
<../tests/08.t
Naive Algorithm ms=1348348

// Тест 2. Текст,состоящий из 10^6 слов,редко повторяющихся.
ivan@Laptop-IM:/mnt/c/Users/Иван/projects/da_labs/da_lab4/benchmark$ ./benchmark_BM
<../tests_02/07.t
Boyer Moore Algorithm ms=1539359
ivan@Laptop-IM:/mnt/c/Users/Иван/projects/da_labs/da_lab4/benchmark$ ./benchmark_BM
<../tests_02/07.t
Naive Algorithm ms=1233279
```

Разбреремся в результатах. Во втором случае сдвиг почти всегда был мизерным из-за отсутствия повторений подстрок, благодаря чему время поиска сильно увеличилось. В первом же случае, уверен, сыграла свою роль эвристика хорошего суффикса: постоянно происходили сдвиги на приличное число шагов.

Тесты создавались с помощью программы на языке Python:

```python
import sys
import random
import string

TEST_COUNT = 8
def get_random_string():
    WORDS = (" cat", " jumble", " easy", " difficult", " answer", " xylophone")
    length = random.randint(0, 1000)
    random_list = [ random.choice(WORDS) for _ in range(length) ]
    return "".join(random_list)
'''
def get_random_string():
    length = random.randint(0, 16)
```

```
    random_list = [ random.choice(string.ascii_letters) for _ in range(length) ]
    return "".join(random_list)
'''
def main():
    if len(sys.argv) != 2:
        print(f"Usage: {sys.argv[0]} <test directory>")
        sys.exit(1)

    test_dir = sys.argv[1]

    lines = [0]
    lines.extend([ 10 ** i for i in range(TEST_COUNT) ])

    for enum, test_count in enumerate(range(1, TEST_COUNT+1)):
        test = []

        line_count = lines[enum]
        for _ in range(line_count):
            value = get_random_string()
            test.append(value)

        test_name = "{}/{:02d}".format(test_dir, test_count)
        with open(f'{test_name}.t', 'w') as ftest:
            for value in test:
                ftest.write(f'{value}\n')
main()
```

4 Выводы

Выполнив четвёртую лабораторную работу по курсу «Дискретный анализ», я изучил алгоритмы поиска образца в строке: алгоритм Кнута-Морриса-Пратта, алгоритм Апостолико-Джанкарло, алгоритм Ахо-Корасик и реализовал алгоритм Бойера-Мура. Алгоритм Бойера-Мура на хороших данных довольно быстр, а вероятность появления плохих данных крайне мала. Поэтому он оптимален в большинстве случаев, когда нет возможности провести предварительную обработку текста, в котором проводится поиск (часто используется в браузерах и текстовых редактора для поиска определенного слова). Но на искусственно подобранных неудачных текстах скорость алгоритма Бойера-Мура серьёзно снижается. Из недостатков можно отметить, что алгоритм Бойера-Мура не расширяется до приблизительного поиска, поиска любой строки из нескольких.

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] Ден Гасфилд. *Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология* — Издательский дом «Невский Диалект», 2003. Перевод с английского: И. В. Романовский. — 654 с. (ISBN 5-7940-0103-8)

[3] *Boyer Moore Algorithm for Pattern Searching*
URL: `https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/` (дата обращения: 21.12.2020).

[4] *Chrono in C++*
URL: `https://www.geeksforgeeks.org/chrono-in-c/` (дата обращения: 02.12.2020).