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: М. А. Волков
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б
Дата: 14 декабря 2020 г.
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 Апостолико-Джанкарло.

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

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

Как сказано в [1]: «Апостолико и Джанкарло предложили вариант алгоритма Бойера-Мура, который допускает замечательно простое доказательство линейной оценки наихудшего времени счета».
Из этого можно сделать вывод, что алгорим «Апостолико-Джанкарло» или также известный, как «Турбо-алгоритм Бойера Мура» (далее ТБМ) ничем практически не отличается от алгоритма поиска Бойера Мура. За исключением одного факта: ТБМ использует дополнительный массив размером с весь текст, что, конечно, не является большим плюсом. Но благодаря этому массиву данный алгоритм проигрывает в ассимптотике в худшем случае. Бойер Мур и ТБМ в среднем работают за $O(n+m)$, где n – количество символов в тексте, m – количество символов в паттерне. Но худшее время работы у Бойера Мура составляет $O(n*m)$, когда как у ТБМ $O(2n)$, что является линейной ассимптотикой.

2 Исходный код

Задача сделана в одном файле для простоты отсылки на чеккер.
В этом файле есть класс $TTBM$, в котором написаны все нижеописанные методы. Этот нестандартный подход к задаче был сделан потому, что, создавая функции, очень большое количество параметров нужно передавать внутрь этой функции. Я очень часто путал какие именно параметры куда и где я использовал. После этого решено было мной сделать один большое контейнер, такой как класс, куда я помещу все часто используемые мной переменные.

Еще несколько плюсов от того, что я реализацию всего задания поместил в один класс состоит в том, что очень здорово разгрузилась функция *main*. Все процессы, которые должны произойти до начала использования алгоритма, происходят в конструкторе класса. Благодаря этому достаточно просто создать объект, создать *vector* ответов и применить к ветору ответов метод *Do* объекста класса $TTBM$.

```
1 int main() {
2   TTBM tbm;
3   std::vector<TAnsVal> ans;
4   ans = tbm.Do();
5   for (auto i : ans) {
6     std::cout << i << std::endl;
7   }
8   return 0;
9 }
```

main.c	
void PatternParser()	Метод класса $TTBM$, который считывает из stdin паттерн.
void TextParser()	Метод класса $TTBM$, который считывает из stdin текст.
static std::vector<int> z(std::vector<unsigned> pat)	Статический метод класса $TTBM$, который считает z-функцию от какого-то паттерна.
TTBM()	Конструктор класса $TTBM$, который выполняет парсинг паттерна и текста и делает все подготовительные задачи с паттерном.
void RoolBadChar()	Метод класса $TTBM$, который реализует ППС.
void RoolGoodSuff()	Метод класса $TTBM$, который реализует ПХС.

std::vector<TAnsVal> Do()	Метод класса $TTBM$, который реализует алгоритм Апостолико-Джанкарло, также известного, как Турбо алгоритм Бойера Мура.

```
1  struct TValue {
2    unsigned val;
3    unsigned int lineCount;
4    unsigned int wordCount;
5  };
6
7  struct TAnsVal {
8    unsigned lineCount;
9    unsigned wordCount;
10 };
11
12 class TTBM {
13 private:
14   std::unordered_map<unsigned, int> rightIdPattern;
15   std::vector<unsigned> Pattern;
16   std::vector<TValue> text;
17   std::vector<int> arrGsN;
18   std::vector<int> M;
19   std::string str;
20
21
22   void PatternParser();
23   void TextParser();
24   static std::vector<int> Z(std::vector<unsigned> pat);
25 public:
26   TTBM();
27   void RoolBadChar();
28   void RoolGoodSuff();
29   std::vector<TAnsVal> Do();
```

3 Консоль

```
11 45 11 45 90
0011 45 011 0045 11 45 90    11
45 11 45 90
^D
1,3
1,8

Process finished with exit code 0
```

4 Тест производительности

Тут Вы описываете собственно тест производительности, сравнение Вашей реализации с уже существующими и т.д.

Тест производительности представляет из себя следующее: производится сравнение между ТМБ и обычным наивным поиском, который реализован через 2 for.
В **тесте 1** находится 10^5 строк и 10 элементов в каждой строке. Паттерн 4 элемента

В **тесте 2** находится 10^5 строк и 10 элементов в каждой строке. Паттерн 200 элементов

В **тесте 3** находится 10^5 строк и 10 элементов в каждой строке. Паттерн 400 элементов

```
test1
Naive: 0 ms
TBM: 0.006 ms

test2
Naive: 0.005 ms
TBM: 0.002 ms

test3
Naive: 0.004 ms
TBM: 0.001 ms
```

Можно заметить, что тесты отличаются только количеством элементов в паттерне. Сразу же можно заметить, что на первом тесте выигрывает наивный поиск. Но чем больше становится паттерн, тем дольше начинает работать наивный поиск. Оно не удивительно, ведь его сложность $O(n*m)$, где n – размер текста, m – размер паттерна.

5 Выводы

Выполнив четвертую лабораторную работу по курсу «Дискретный анализ», я узнал о разных алгоритмах поиска подстроки в строке и на практике научился использовать один из них. Также понял, что очень редко встречаются худшие случаи для алгоритма Бойера Мура, поэтому резонность использования, так называемого, турбо сдвига, учитывая потребление памяти, очень мала.

Список литературы

[1] Дэн Гасфилд. *Строки, деревья и последовательности в алгоритмах* — Невский Диалект БЧВ-Петербург, 2003. Перевод с английского: И. В. Романовского

[2] *Бойер Мур — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Алгоритм_Бойера_-_Мура` (дата обращения: 14.12.2020).

[3] *Турбо-алгоритм Бойера Мура — Вики Итмо.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Турбо-алгоритм_Бойера-Мура` (дата обращения: 14.12.2020).

[4] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008