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: Н. И. Лохматов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б
Дата: 26.05.2023
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №4

Задача:

Вариант №6-1

Вариант алгоритма: Поиск одного образца-маски: в образце может встречаться «джокер» (представляется символом ? - знак вопроса), равный любому другому символу. При реализации следует разбить образец на несколько, не содержащих «джокеров», найти все вхождения при помощи алгоритма Ахо-Корасик и проверить их относительное месторасположение.

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. 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.

Примеры

Входные данные

```
cat ? dog
cat
cat
dog
dog
```

Результат

```
1, 1
2, 1
```

1 Описание

Для решения задачи построим trie для входной строчки, примем за букву отдельные слова, знаки вопроса (Джокеры) считаются целыми словами. Используем препроцессинг считав всю строчку целиком, разобьем её на куски, в которых содержатся только буквы, или только Джокеры, отправим данный массив строк в конструктор для trie. В свою очередь trie, получая новое слово состоящее из букв, будет начиная от корня добавлять это слово, проходя по уже существующим вершинам, или добавляя новые, после добавления всех слов вызывается функция добавления суффиксных связей, что позволит нам в дальнейшем дойдя до конкретной точки, сразу переместиться в нужную для нас вершину, которая является суффиксом но на единичку короче нашей ветки, таким образом, не найдя путь дальше, мы сможем продожить поиск, не возвращаясь по тексту обратно на длину глубины в боре. Что делает данный алгоритм онлайн алгоритмом, при условии, что мы заранее узнали целый паттерн по которому мы будем искать в тексте слова, ведь при анализе текста мы уже не сможем достроить наш trie.

2 Исходный код

Сложность алгоритма - $\Theta(m+n+k)$, где

m - суммарная длина паттернов n - построение trie, k - количество вхождений всех паттернов.

main.cpp	
void ToLower(char &ch)	Приведение символов к нижнему регистру.
struct TTrieNode	Структура ноды дерева.
void TTrieNode::AddWord(vector <string> &word, int gShift, int pos)	Добавляем слово в ноду.
void TTrieNode::LinkSuffix()	Строим суффиксные ссылки.
void TTrieNode::LinkExit()	Строим сслыки выхода.
class TTrie()	Класс дерева.
void TTrie::AddWord(vector <vector <string> > &vPattern	Добавляем слово в дерево.
class TText()	Класс вводимого текста.
void TAhoKorasik::InsertTrie(vector <vector <string> > &vPattern)	Вставка в дерево.
void TAhoKorasik::Search()	Поиск в дереве.
void Parser(string &str, vector<string> &words)	Парсер паттерна.
void Clear(vector <string> &pattern, vector <vector <string> > &vPattern, vector <string> &words, bool flag)	Очистка векторов.

Структура дерева:

```
1   class TTrie {
2   public:
3       TTrie () : gShift(-1) {
4           root = new TTrieNode();
5       }
6       ~TTrie ();
7       void AddWord(vector <vector <string> > &vPattern);
8       void LinkTrie();
9       TTrieNode * root;
10      int gShift;
11      int wordCount;
12  };
```

Структура ноды дерева:

```
1   struct TTrieNode {
```

```
2       unordered_map <string, TTrieNode *> next;
3       bool isLeaf;
4       TTrieNode *parent;
5       TTrieNode *suffix;
6       TTrieNode *exit;
7       string patCH;
8       vector <int> shift;
9
10      TTrieNode() : isLeaf(false), parent(nullptr), suffix(nullptr), exit(nullptr) {}
11      ~TTrieNode();
12      void AddWord(vector <string> &word, int gShift, int pos);
13      void LinkSuffix();
14      void LinkS();
15      void LinkExit();
16      void LinkE();
17  };
```

3 Консоль

```
separatrix@separatrix-virtual-machine:~/Desktop/da labs$ cat test.a
? ?
r y i x n
j i i l
s
separatrix@separatrix-virtual-machine:~/Desktop/da labs$ ./lab4 <test.a
1,1
1,2
1,3
1,4
1,5
2,1
2,2
2,3
2,4
```

4 Тест производительности

Тесты производительности представляют из себя следующее: алгоритм Ахо-Корасика будет сравниваться с поиском с помощью наивного алгоритма, ипользуя маски (за линейное время). Входной файл - строка с паттерном длиной 100 слов в каждом. Текст представляет собой от 1000 до 10000000 строк.

```
Alg:
pattern 100 text 1000 0.0543351173401
pattern 100 text 10000 0.394222974777
pattern 100 text 100000 3.83085799217
pattern 100 text 1000000 38.2940719128
pattern 100 text 10000000 396.297783852

AH:
pattern 100 text 1000 0.00444197654724
pattern 100 text 10000 0.0231420993805
pattern 100 text 100000 0.183077096939
pattern 100 text 1000000 2.38923597336
pattern 100 text 10000000 18.7939031124
```

Как видно, Ахо-Корасик работает заметно быстрее.

5 Выводы

Выполнив четвёртую лабораторную работу по курсу «Дискретный анализ», я на практике закрепил знания об алгоритме Ахо-Корасика, которыпозволяет линиализировать поиск шаблона по тексту, что дает большие возможности, также порождая другие возможные использования данного алгоритма. Данная работа оказалась самой сложной для меня за семестр, хотя её выплнение заняло у меня меньше времени, чем 2-3.

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.))

(дата обращения; 19.03.20).

[2] *aho corasick*
URL: https://e-maxx.ru/algo/ahocorasick

(дата обращения; 19.03.20).

[3] *aho corasick*
URL: http://aho-corasick.narod.ru/

(дата обращения: 19.03.20).