Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Дискретный анализ»

Студент: И. П. Попов
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б

Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №5

Задача:

Найти в заранее известном тексте поступающие на вход образцы.

Формат ввода

Текст располагается на первой строке, затем, до конца файла, следуют строки с образцами.

Формат вывода

Для каждого образца, найденного в тексте, нужно распечатать строчку, начинающуюся с последовательного номера этого образца и двоеточия, за которым, через запятую, нужно перечислить номера позиций, где встречается образец в порядке возрастания.

1 Описание

Алгоритм Укконена -это линейно-временной онлайн-алгоритм построения деревьев суффиксов, предложенный Эско Укконеном в 1995 году. Алгоритм начинается с неявного дерева суффиксов, содержащего первый символ строки. Затем он проходит по строке, добавляя последовательные символы до тех пор, пока дерево не будет завершено.

Класс суффиксного дерева, для детерменированности описания состояния, в котором оно нахоодится в лююой момент времени, имеет следующие параметры:
активная точка представляет тройку (active node, active edge, active len)
remainder представляет собой количество новых суффиксов, которые нужно вставить

Основные правила, которые используются при добавлении в дерево нового символа:
Правило 1, после вставки из корня:
- active node остается корнем
- active edge становится первым символом нового суффикса, который нужно вставить, т.е. b
- active len уменьшается на 1

Правило 2
Если ребро разделяется и вставляется новая вершина, и если это не первая вершина, созданная на текущем шаге, ранее вставленная вершина и новая вершина соединяются через специальный указатель, суффиксную ссылку.

Правило 3
После разделения ребра из active node, которая не является корнем, переходим по суффиксной ссылке, выходящей из этой вершины, если таковая имеется active node устанавливается вершиной, на которую она указывает. Если суффиксная ссылка отсутствует, active node устанавливается корнем.active edge и active len остаются без изменений.

2 Исходный код

Применение вышеописанных правил на практике происходит в функции добавления символа в дерево:

```
1  void TSuffixTree::AddLetter(string::iterator add){
2      lastAdded = root;
3      ++remainder;
4      while (remainder) {
5          if (activeLen == 0) {
6              activeEdge = add;
7          }
8
9          map<char, TNode *>::iterator it = activeNode->to.find(*activeEdge);
10         TNode *next;
11
12         if (it == activeNode->to.end()) {
13             TNode *leaf = new TNode(add, text.end());
14             activeNode->to[*activeEdge] = leaf;
15
16             if (lastAdded != root) {
17                 lastAdded->sufflink = activeNode;
18             }
19             lastAdded = activeNode;
20
21         }
22
23         else {
24             next = it->second;
25             if (CheckEdge(add, next)) {
26                 continue;
27             }
28
29             if (*(next->begin + activeLen) == *add) {
30                 ++activeLen;
31                 if (lastAdded != root) {
32                     lastAdded->sufflink = activeNode;
33                 }
34
35                 lastAdded = activeNode;
36                 break;
37             }
38
39             TNode *split = new TNode(next->begin, next->begin + activeLen);
40             TNode *leaf = new TNode(add, text.end());
41             activeNode->to[*activeEdge] = split;
42
43             split->to[*add] = leaf;
```

```
            next->begin += activeLen;
            split->to[*next->begin] = next;


            if (lastAdded != root){
                lastAdded->sufflink = split;
            }

            lastAdded = split;
        }


        --remainder;
        if (activeNode == root && activeLen) {
            --activeLen;
            activeEdge = add - remainder + 1;

        }

        else {
            activeNode = activeNode->sufflink;
        }
    }
}
```

3 Консоль

tmp:

```
abcdabc
abcd
bcd
bc
```

console output:

```
1: 1
2: 2
3: 2,6
```

4 Тест производительности

Тест производиительности представляет собой сравнение реализованного мной алгоритма Укконена и STL find.

Сравнение производится путем запуска поиска 400000 паттернов в тексте.

В результате работы *benchmark.cpp* видны следующие результаты:

```
root@Lunidep:~/DA/da_lab5# ./wrapper.sh
[info][Mon Oct  3 22:29:56 MSK 2022] Stage #1. Compiling...
g++ -std=c++11 -pedantic -Wextra -Wall -Werror -Wno-sign-compare -Wno-unused-result
-o da_lab5 TSuffixTree.cpp main.cpp
g++ -std=c++11 -pedantic -Wextra -Wall -Werror -Wno-sign-compare -Wno-unused-result
-o benchmark TSuffixTree.cpp benchmark.cpp
[info][Mon Oct  3 22:29:58 MSK 2022] Compiling OK
[info][Mon Oct  3 22:29:58 MSK 2022] Stage #2. Benchmark generating...
[info][Mon Oct  3 22:29:58 MSK 2022] Benchmark generating OK
[info][Mon Oct  3 22:29:58 MSK 2022] Stage #3. Program testing...
1: 1
2: 2
3: 2,6
[info][Mon Oct  3 22:29:58 MSK 2022] Program testing OK
[info][Mon Oct  3 22:29:58 MSK 2022] Stage #4. Benchmark results:
Time for create suffix tree: 67226 ms
Time for my find: 96589162 ms
Time for standart find: 196576595 ms
[info][Mon Oct  3 22:29:59 MSK 2022] Benchmark OK
```

5 Выводы

Выполнив пятую лабораторную работу по курсу «Дискретный анализ», мною были изучены суффиксные деревья, а также различные виды их построения и использования для решения прикладных задач.

С их помощью, обработав текст можно получить поиск сложностью O(длина текста), в то время, как у алгоритмов, разобранных в 4 лабораторной работе сложность поиска составляла O(длина паттерна). В случае достаточного количества памяти и понимания, что длины паттернов » длина текста, предпочтительнее использовать Суффиксное дерево, нежели пользоваться другими способами поиска подстроки в строке.

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.))