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: Ф. А. Иванов
Преподаватель: И. Н. Симахин
Группа: М8О-208Б-19

Дата: 21.04.2022
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №5

Задача: Необходимо реализовать алгоритм Укконена построения суффиксного дерева за линейное время. Построив такое дерево для некоторых из выходных строк, необходимо воспользоваться полученным суффисным деревом для решения своего варианта задания.

Алфавит строк: строчные буквы латинского алфавита (т.е. от a до z).

Вариант: Найти в заранее известном тексте поступающие на вход образцы.

Формат входных данных: Текст располагается на первой строке, затем, до конца файла, следуют строки с образцами.

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

1 Описание

Суффиксное дерево позволяет производить поиск подстроки в строке за O(m+k), где m это длина подстроки, а k число вхождений. Для быстрой работы нам также необходимо быстро строить само суффиксное дерево. В этом нам помогает флгоритм Укконена, который строит суффиксное дерево за линейное время относительно длиня текста.

Начнем с построения суффиксного дерева. Суффиксное дерево будет на основе com pact trie, а вместо строк в ребрах будут использоваться номера индексов начала и конца участка в исходном тексте. Trie будет достраиться для каждого символа текста. Добавление символов состоит и нескольких этапов. Во первых мы добавляем наш символ ко всем ребрам листьям, так как они являлись концами суффиксов строки без нашего нового символа.

Теперь нам нужно добавить все не достающие суффиксы, при этом мы уже на какой-то позиции в trie, мы можем оказаться в одной из ситуаций, символ уже существует (то есть наш суффикс является подстрокой другого), мы между ребрами и ребра, который начинается с нашего символа не существует, мы посреди ребра и следующий символ отличается от нашего. В первом случае мы просто переходим в trie по этому символу и закончить, во втором случае мы должны должны создать новый лист с нужным символом, а в третьем нам придется разорвать ребро на два и уже потом создать новый лист.

Во втором и третьем случае мы должны продолжать цикл до того момента пока мы не окажемся либо в корне после выполнения, либо пока не попадем в первый слечай. При этом мы для повтора каждый раз переходим по суффиксной ссылке или если ребро начинается из корне, переходим по суффиксу ребра.

Для суффксных создания мы при каждом разделении ребра по третьему случаю, записываем указатель на место разделения в отдельюную переменную, а при последующий создании чего-то нового по правилам 2 и 3 проверяем нужно ли нам привязать эту суффиксную ссылку к аналогичному символу

2 Исходный код

```
class SUFTree {
public:
    int countSuffix = 0;
    int countList = 0;
    int position = 0;
    int currentEdge;

    struct Node;

    Node *root;
    Node *currentNode;
    Node *lastAdd;

    string inputString;

    void addNode(int pos);
    void createList(int pos, Node *node);
    void buildSufflink(Node *node);
    void walkSufflink();
    void breakNode(int pos);
    bool edgeFault();


    SUFTree(string &inputString);
    ~SUFTree();
    void solution(string &pattern, vector<int> &answer);
};

struct SUFTree::Node
{
    int numberList;
    int left, right;

    Node *Sufflink = nullptr;
    map<char, Node *> edges;
    Node (int left, int right, int numberList) {
        this->left = left;
        this->right = right;
        this->numberList = numberList;
    }
    ~Node();
    void listsNumbers(vector<int> &answer);
};
```

3 Консоль

```
orion@orion-laptop:~/University/DA/Lab_05/solution$ g++ main.cpp -o main
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main
abcdabcdbc
ab
1: 1,5
abc
2: 1,5
abcd
3: 1,5
cdb
5: 7
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main
abcdabc
abcd
1: 1
bcd
2: 2
bc
3: 2,6
```

4 Тест производительности

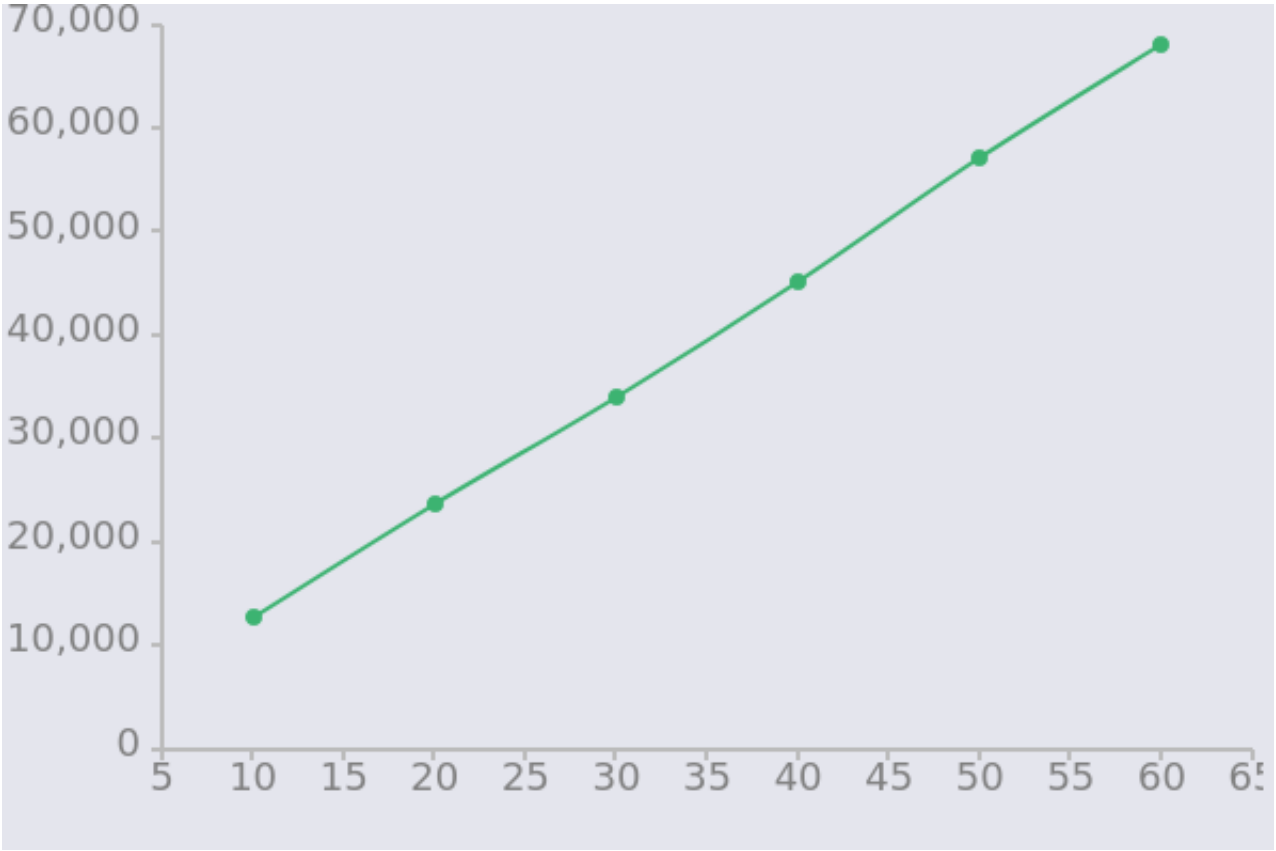
```
orion@orion-laptop:~/University/DA/Lab_05/solution$ g++ main.cpp -o main
orion@orion-laptop:~/University/DA/Lab_05/solution$python3 creater.py
Количество символов: 10
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main <test
Время построения: 104
orion@orion-laptop:~/University/DA/Lab_05/solution$python3 creater.py
Количество символов: 20
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main <test
Время построения: 218
orion@orion-laptop:~/University/DA/Lab_05/solution$python3 creater.py
Количество символов: 30
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main <test
Время построения: 340
orion@orion-laptop:~/University/DA/Lab_05/solution$python3 creater.py
Количество символов: 40
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main <test
Время построения: 401
orion@orion-laptop:~/University/DA/Lab_05/solution$python3 creater.py
Количество символов: 50
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main <test
Время построения: 522
orion@orion-laptop:~/University/DA/Lab_05/solution$python3 creater.py
Количество символов: 60
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main <test
Время построения: 629
orion@orion-laptop:~/University/DA/Lab_05/solution$
```

Здесь и без графика видна линейная сложность построения дерева. Теперь посмотрим на нахождение за линию. Количество тестов на нахождению за линию будет константным = 5. Будем изменять длину строки.

```
orion@orion-laptop:~/University/DA/Lab_05/solution$ g++ main.cpp -o main
orion@orion-laptop:~/University/DA/Lab_05/solution$python3 creater.py
Количество символов: 10
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main <test
Время построения: 12742
orion@orion-laptop:~/University/DA/Lab_05/solution$python3 creater.py
Количество символов: 20
```

```
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main <test
Время построения: 23770
orion@orion-laptop:~/University/DA/Lab_05/solution$python3 creater.py
Количество символов: 30
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main <test
Время построения: 34101
orion@orion-laptop:~/University/DA/Lab_05/solution$python3 creater.py
Количество символов: 40
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main <test
Время построения: 45128
orion@orion-laptop:~/University/DA/Lab_05/solution$python3 creater.py
Количество символов: 50
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main <test
Время построения: 57192
orion@orion-laptop:~/University/DA/Lab_05/solution$python3 creater.py
Количество символов: 60
orion@orion-laptop:~/University/DA/Lab_05/solution$ ./main <test
Время построения: 68145
orion@orion-laptop:~/University/DA/Lab_05/solution$
```

Исходя из данных мы видем линейную зависимовсть поиска подстроки в строке.


70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
5
10
15
20
25
30
35
40
45
50
55
60

5 Выводы

Выполнив данную лабораторную работу, я познакомился с суффиксными деревьями. Его основое предназначение поискобразцов в тексте. В отличии от расмотренных в прошлом семестре алгоритмов поиска образцов в тексе, суффиксное дерево отличается тем, что преобразовывает текст, а не образец.

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.))