

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

**(ДВФУ)**

|  |
| --- |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**  **Департамент математического и компьютерного моделирования** |

**ДОКЛАД**

**о практическом задание по дисциплине АИСД**

«Алгоритм Укконена»

направление подготовки 09.03.03 «Прикладная информатика»

профиль «Прикладная информатика в компьютерном дизайне»

|  |  |  |
| --- | --- | --- |
|  |  | Выполнил студент  гр. Б9121-09.03.03пикд  Талалуев Денис Сергеевич \_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Доклад защищен:  С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | *(подпись)*  Руководитель практики  Доцент ИМКТ А.С Кленин  *(должность, уч. звание)*  \_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)*  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2022г. |
| Рег. № \_\_\_\_\_\_  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. |  |  |

г. Владивосток

2022Содержание

[Описание алгоритма 6](#_Toc124849242)

Введение

Алгоритм Укконена – это алгоритм построения дерева суффиксов с линейным временем O(n). Алгоритм начинается с неявного дерева суффиксов, содержащего первый символ строки. Затем он проходит по строке, добавляя последовательные символы, пока дерево не будет завершено. Суффиксное дерево, построенное алгоритмом Укконена, владеет рядом характеристик, из-за которых дерево строиться за линейное время и занимает минимальное количество памяти.

Данное исследование предназначено для построения доступной и понятной документации по данному алгоритму. В этой работе будут освещены все разновидности, минусы и плюсы алгоритма, а также его реализация на практике с описанием каждого шага построения дерева суффиксов.История алгоритма

Эско Укконен придумал данный алгоритм в 1977 году Оригинальный алгоритм, представленный П[итером Вайнером](https://translated.turbopages.org/proxy_u/en-ru.ru.cb165c08-633a22aa-8f98b1a3-74722d776562/https/en.wikipedia.org/w/index.php?title=Peter_Weiner&action=edit&redlink=1), проходил в обратном порядке от последнего символа к первому, от самого короткого к самому длинному суффиксу. Более простой алгоритм был найден [Эдвардом М. Маккрейтом](https://translated.turbopages.org/proxy_u/en-ru.ru.cb165c08-633a22aa-8f98b1a3-74722d776562/https/en.wikipedia.org/wiki/Edward_M._McCreight), переходящим от самого длинного к самому короткому суффиксу.Особенности алгоритма

Данный алгоритм строит суффиксное дерево (бор), в котором находятся все подстроки данной нам строки. При построении дерева мы записываем каждый символ строки в вершину нашего дерева, но алгоритм Укконена превращает обычное суффиксное дерево в сжатое при помощи сжатия множества ребер без развилок в одно целое, в котором записана подстрока. При сжатии ребер мы не записываем целую подстроку в ребро, а записываем индексы начала и конца подстроки. Это обусловлено тем что запись целой подстроки занимает большое количество памяти, а точнее O(n2), но при записи индексов затрата памяти сокращается до O(n).

Для быстрого построения суффиксного дерева в алгоритме Укконена используются суффиксные ссылки. Суффиксная ссылка вершины v называется указатель на вершину u, такую что стока u – является наибольшим суффиксом строки v, который уже есть в дереве (строку считаем от корня до самой вершины), а есть такой вершины не существует ссылка указывает на корень дерева.

# Описание алгоритма

Алгоритм читает строку слева направо. Каждый шаг может включать в себя больше, чем одну операцию, но в итоге общее количество операций будет O(n).

Базовый принцип алгоритма

В этой части будет продемонстрирована работа алгоритма с самой простой строкой “abc”. Цель алгоритма – на каждом шаге продлевать существующие суффиксы и создавать новые, путем добавления символов заданной строки.

Алгоритм начинается с символа “a”, в дерево вставляется ребро из корня к листу и помечается как [0, #], где # - текущий конец подстроки. Это означает что ребром является подстрока, которая начинается с позиции корня до # (в данный момент текущим концом является позиция 1)

Это выглядит так:

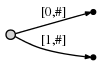


это означает



Далее алгоритм переходит на следующий символ “b” и увеличивает # на один. Поэтому на данном шаге вставляется новое ребро “b” и продлевается существующее ребро до “ab”.

Сейчас это выглядит так:



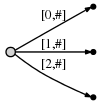
это означает



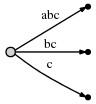
Так как # увеличился, то существующее ребро [0, #] тоже автоматически продлилось до символа с индексом #.

На последнем шаге происходит то же самое что и на предыдущем (обновляются существующие ребра и добавляется одно новое).

Таким образом получается такое дерево.



это означает



Алгоритм завершился и выполнил свою работу за O(n), это обуславливается тем, что на каждом шагу не приходилось переписывать существующие ребра, а они просто продлевались с помощью # (текущего конца).

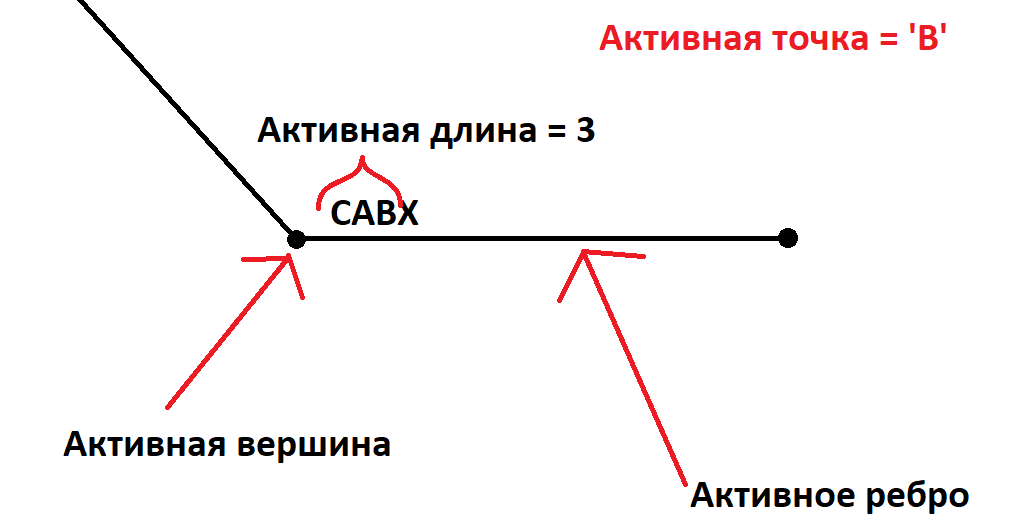
Из-за того, что символы в данной строке ни разу не повторились, алгоритм сработал самым простым способом, каждый шаг добавлял новое ребро и продлевал существующие. Таким образом ни одна вершина кроме корня не имеет дочерних элементов.

Расширение алгоритма.

В данной части будет рассмотрена работа алгоритма со строкой, у которой повторяются символы.

Чтобы это сделать нужно ввести новые переменные, которые не использовались в прошлом примере (активная точка и остаток).

Активная точка – переменная, которая состоит из трех значений (активная вершина, активное ребро, активная длина). Активная точка — это такая точка, которая может находиться по середине какого-то ребра, с помощью нее будет проводится построение дерева таким образом чтобы это затрачивало как можно меньше времени. Активная вершина — это точка, из которой проходит ребро, в котором находится точка. Активное ребро – это ребро, на котором находится точка, оно обозначается первой буквой подстроки этого ребра, так как они не повторяются. Активная длина – расстояние, на котором находится точка от вершины по ребру.



Остаток – это количество суффиксов, которое нужно вставить, на каждом шагу остаток увеличивается на 1, а при установке суффикса уменьшается на 1

Рассмотрим строку “abcabxabcd”.

Первые три шага выполняются в точности с прошлым примером.

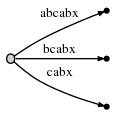
Для быстрого и правильного построения дерева алгоритм каждый шаг делает проверку на повторение символов, если они повторяются, то они входят в очередь на вставку до тех пор, пока не встретится символ, который еще не встретился, после этого начинается вставка суффиксов (этот случай описан в шестом шаге).

Четвертым шагом при в ставке следующего символа “a” (“abc**a**bxabcd”) из-за того, что этот символ уже присутствует в ребре, он не вставляется. Активная точка устанавливается как (корень, ‘a’, 1), а остаток увеличивается на 1, то есть становится равным 2.

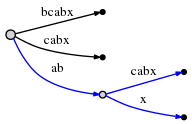
На пятом шагу при вставке символа “b” (“abca**b**xabcd”), алгоритм переходит к активной точке и опять этот символ присутствует на этом же ребре, поэтому устанавливается активная точка как (корень, ‘a’, 2) и приращивается остаток до 3.

Замечание: так как текущий конец увеличивается на каждом шаге тогда и все суффиксы, которые не были установлены, тоже продлеваются, как остальные суффиксы, которые были уже вставлены в ребра.

На шестом шагу алгоритм встречает символ “x” (“abcab**x**abcd”), из-за того, что он еще ни разу не встречался алгоритм начинает вставлять все суффиксы, находящиеся в очереди с первого по последний (в ходе вставки суффиксов алгоритм вставляет только один символ ‘x’, так как начало суффиксов уже находятся в дереве). В данный момент очередь на вставку состоит из суффиксов “abx”, “bx”, “x”.



Активная точка находится на ребре “abcabx” на второй позиции на символе ‘b’,тоесть (корень, ‘a’, 2). Алгоритм разделяет данное ребро на 2 после символа ‘b’ и вставляет символ “x”



Остаток уменьшается на 1 и алгоритм переходит к установке следующего суффикса “abx”. После каждой вставки суффикса путем разделения ребра алгоритм меняет активную точку по правилам, которые изложены по ходу отчета.

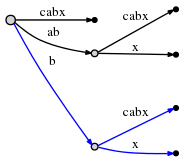
Правило 1: если после разделения дерева, активная вершина является корнем, то алгоритм работает следующим образом.

- активная вершина остается корнем

- активное ребро становится первым символом следующего суффикса в очереди, который нужно вставить, т. е. b

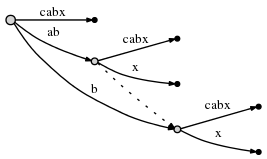
- активная длина уменьшается на 1

Следовательно, новая тройка активной точки (корень, 'b', 1) указывает, что следующая вставка должна быть сделана на ребре “bcabx”, за 1 символом. После этого алгоритм разделяет ребро на два и вставляет новый символ.

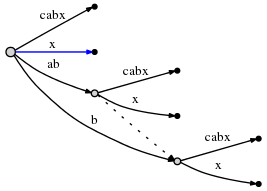


Далее активная точка изменяется согласно правилу 1 на (корень, 'x', 0) и в работу включается правило 2.

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 (суффиксная ссылка на изображении указана пунктирной линией).**



Последнее что делает, алгоритм на данном шаге он вставляет последний символ согласно активной точке, то есть добавляет новое ребро от корня дерева, потому что в дереве нет ребра, которое начинается на ‘x’.



На следующих трех шагах алгоритм добавляет суффиксы в очередь согласно предыдущим шагам так как они уже присутствуют в ребрах и изменяет активную точку.

На 7 шагу на символе “a” (“abcabx**a**bcd”) алгоритм добавляет суффикс в очередь и изменяет активную точку как (корень, ‘а’, 1).

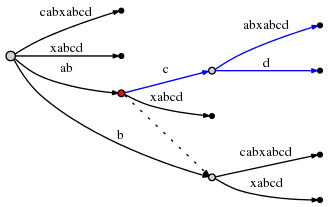
На 8 шагу на символе “b” (“abcabxa**b**cd”) суффикс добавляется в очередь, и активная точке меняется на (корень, ‘a’, 2), но из-за того, что точка в этом случае указывает на последний символ на ребре, алгоритм меняет активную точку на следующую вершину, то есть (вершина1, ‘\0x’, 0).

В тексте вершина указана как “вершина1”, но на практике все вершины в дереве должны быть проиндексированы.

На 9 шагу на символе “c” (“abcabxab**c**d”) в очередь добавляется новый суффикс в очередь и изменяет активную точку на (вершина1, ‘c’, 1).

По итогу после этих трех шагов в очереди лежат три суффикса (“abc”, “bc” и “c”)

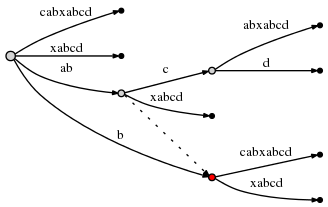
На последнем шаге алгоритм встречает символ ‘d’ (“abcabxabc**d**”), который не встречается ни на одном ребре активной точки и начинает вставлять все суффиксы из очереди по порядку. Сначала вставляется суффикс “abcd”, путем добавления символа ‘d’.



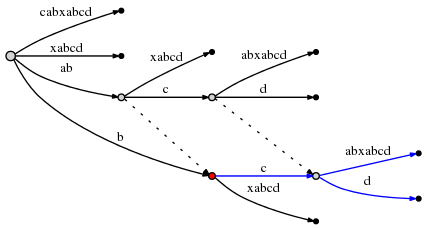
Далее в работу вступает последнее правило 3.

Правило 3: после разделения ребра из активная вершина, которая не является корнем, переходим по суффиксной ссылке, выходящей из этой вершины, если таковая имеется активная вершина устанавливается вершиной, на которую она указывает. Если суффиксная ссылка отсутствует, активная вершина устанавливается корнем. Активное ребро и активная длина остаются без изменений.

После этого активная точка меняется на (вершина 2, ‘c’, 1)

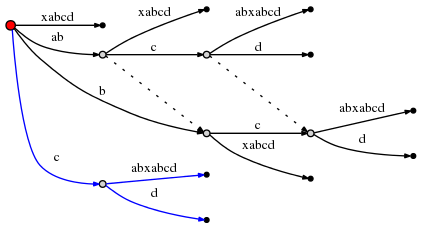


Далее алгоритм добавляет новый суффикс согласно активной точке и создает суффиксную ссылку по правилу 2.



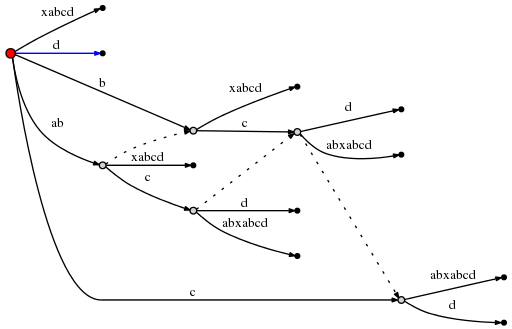
Из-за того, что данная вершина не имеет суффиксной ссылки по правилу 3 активная вершина меняется на корень. Теперь активная точка (корень, ‘c’, 1).

Далее вставляется новый суффикс согласно активной точке и создается суффиксная ссылка.



Далее следуем правилу 2и устанавливаем новую суффиксную ссылку из ранее созданной внутренней вершины.

Так как активная вершина является корневой, по правилу 1 активная точка обновляется до (корень, ‘d’, 0). Это означает, что последней вставка текущего шага является вставка единственного d в корень.



Это был последний шаг алгоритма и в итоге получилось явное сжатое дерево суффиксов.

Примечание: если в конце работы алгоритма остаток > 0 ,это может случиться когда работа алгоритма заканчивается на символе, который повторяется в строке, тогда для правильной работы в самом начале к стоке добавляется символ ‘$’ (главное значение этого символа – это его индивидуальность, он не должен встречать больше в строке), так как он отличен от всех остальных символов, алгоритм завершит свою работу правильно.

Источники

1. <https://translated.turbopages.org/proxy_u/en-ru.ru.8fb2c6b6-633a17c2-e5269a51-74722d776562/https/en.wikipedia.org/wiki/Ukkonen's_algorithm>
2. <https://habr.com/ru/post/533774/>
3. <https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Укконена>
4. <https://ru.wikipedia.org/wiki/Суффиксное_дерево#Неявные_суффиксные_деревья>.
5. <https://translated.turbopages.org/proxy_u/en-ru.ru.8f7ed41d-633a225e-d9cc880a-74722d776562/https/www.cs.helsinki.fi/u/ukkonen/SuffixT1withFigs.pdf>
6. <https://habr.com/ru/post/198682/>
7. <https://algorithmica.org/ru/trie>
8. <https://codeforces.com/blog/entry/16780?mobile=true>