

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 Маккрейта»

направление подготовки 09.03.03 «Прикладная информатика»

профиль «Прикладная информатика в компьютерном дизайне»

|  |  |  |
| --- | --- | --- |
|  |  | Выполнила студентка  гр. Б9121-09.03.03пикд  Белкова Елизавета Алексеевна  \_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Доклад защищен:  С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | *(подпись)*  Руководитель практики  Доцент ИМКТ А.С Кленин  *(должность, уч. звание)*  \_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)*  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2023г. |
| Рег. № \_\_\_\_\_\_  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г. |  |  |

г. Владивосток

2023

Оглавление

[Глоссарий 3](#_Toc127202880)

[Введение 5](#_Toc127202881)

[Сферы применения алгоритмов построения суффиксных деревьев 6](#_Toc127202882)

[История строковых алгоритмов 7](#_Toc127202883)

[Принцип работы алгоритма 8](#_Toc127202884)

[Основная идея алгоритма 8](#_Toc127202885)

[Схема алгоритма Маккрейта: 9](#_Toc127202886)

[Схема алгоритма Маккрейта с использованием суф. ссылок: 11](#_Toc127202887)

[Заключение 12](#_Toc127202888)

[Список источников 13](#_Toc127202889)

# Глоссарий

**Символ –** это примитивный тип данных, экземпляры которого имеют уникальную удобочитаемую форму.

**Текст –** это последовательность символов определённой длины.

**Суффиксное дерево** — бор, содержащий все суффиксы некоторой строки (и только их). Позволяет выяснять, входит ли строка *w* в исходную строку *t*, за время *O(|w|)*, где *|w|* — длина строки *w*.

**Бор** (англ. *trie*, *луч*, *нагруженное дерево*) — структура данных для хранения набора строк, представляющая из себя подвешенное дерево с символами на рёбрах. Строки получаются последовательной записью всех символов, хранящихся на рёбрах между корнем бора и терминальной вершиной. Размер бора линейно зависит от суммы длин всех строк, а поиск в бору занимает время, пропорциональное длине образца.

**Суффиксный автомат (или ориентированный ациклический граф слов)** — структура данных, позволяющая хранить в сжатом виде и обрабатывать информацию, связанную с подстроками данной строки. На интуитивном уровне, суффиксный автомат можно понимать как сжатую информацию обо всех подстроках данной строки.

**Ориентированный граф (кратко орграф)** — (мульти) [граф](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), рёбрам которого присвоено направление. Направленные рёбра именуются также дугами, а в некоторых источниках и просто рёбрами. Граф, ни одному ребру которого не присвоено направление, называется неориентированным графом или неорграфом.

**Суффиксная ссылка**– это ссылка из вершины, соответствующей слову *S*, в вершину, соответствующую наидлиннейшему суффиксу *S*, присутствующую в автомате.

**Дерево Патриции (Patricia Merkle Tree)** предоставляет собой криптографически аутентифицированную структуру данных, которую можно использовать для хранения всех привязок (ключ, значение).

**Алгоритм МакКрейта** (англ. *McCreight's algorithm*) — алгоритм построения суффиксного дерева для заданной строки *S* за линейное время. Он добавляет суффиксы в порядке убывания длины. По своей сути является улучшенной версией алгоритма Вайнера.

**S** – строка текста, состоящая из символов типа данных char

**suf** – суффикс строки S

**T** – суффиксное дерево

Введение

Одним из способов представления информации являются письменные тексты. Тексты занимают центральное место в системах «обработки текстов», которые предоставляют средства для манипулирования текстами. Такие системы обычно обрабатывают достаточно большие объекты. Например, этот доклад содержит, вероятно, более тысячи символов. Текстовые алгоритмы встречаются во многих областях науки. В биологии текстовые алгоритмы применимы при изучении молекулярных последовательностей. Сложность текстовых алгоритмов также является одной из центральных и наиболее изучаемых проблем теоретической информатики. Можно сказать, что это область, где практика и теория очень близки друг к другу.

Основной текстовой проблемой является проблема, называемая сопоставлением с образцом («pattern matching»). Для компьютерных приложений требуется функция, которая находит определенную подстроку символов в более длинной основной строке, то есть решает проблему «pattern matching». Одним из существующих решений является контекстный поиск в текстовом редакторе. Другие приложения включают автоматическую команду выполнение этого поиска. Функция поиска подстроки также полезна в качестве основы в построении более сложных совпадений шаблонов.

Наивный алгоритм реализации этой функции просто пытается сопоставить подстроку с основной строкой во всех возможных выравниваниях. Это просто, но в некоторых случаях данный алгоритм может быть слишком медленным, так как, например, программа может перепроверить тот факт, что позиция 17 в основной строке является символом a почти так же часто, как количество символов в подстроке (например строка *aaaaaaab*).

В том случае, когда осуществляется поиск в очень длинных строках, то практически все алгоритмы быстрого поиска становятся неприменимы из-за требования большого количества дополнительной памяти. При этом применимы только те из них, которые не требовательны к памяти, но зато они медленнее. Как правило, в таких случаях лучше использовать алгоритмы, в которых скорость поиска осуществляется за линейное время. В основном эти алгоритмы базируются на суффиксных деревьях.

# Сферы применения алгоритмов построения суффиксных деревьев

Задачи быстрого и эффективного поиска возникают в большом количестве приложений. Например, в информационно-поисковых системах библиотечных каталогов, интернет-браузерах, которые просеивают огромные количества текстов в поисках материалов, содержащих данные ключевые слова, в электронных журналах, в обслуживании телефонных справочников, в интерактивных энциклопедиях и т.д.

Поиск строк в текстах, расшифровка ДНК, антиплагиат, алгоритмы сжатия за счет выявления повторений и др. – задачи, требующие быстрого и экономного по затратам решения. *Suffix Tree* и *Suffix Array* (Суффиксное дерево и массив суффиксов) – главные структуры данных, позволяющие решать все эти задачи.

Также суффиксные деревья можно применить и при решении других задач: задача о количестве различных подстрок в данной строке, задача о построении суффиксного автомата, задача о поиске наибольшей подстроки двух строк.

Но суффиксные деревья применяются не только в информатике, они также могут эффективно применяться и в молекулярной биологии для работы с молекулярными последовательностями (строками), где эффективность обработки информации играет не последнюю роль.

# История строковых алгоритмов

Первый асимптотически эффективный строковый алгоритм был открыт Кнутом, Праттом и, независимо от них, Моррисом в 1970 г. Он включает в себя предварительную обработку подстроки в автомате поиска, а затем передачу основной строки в автомат поиска, по одному символу за раз. В обоих этих алгоритмах среднее время работы алгоритма линейно зависит от объёма входных данных, то есть разработать [асимптотически более эффективный](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) алгоритм невозможно. Результаты своей работы они опубликовали совместно в [1977 году](https://ru.wikipedia.org/wiki/1977_%D0%B3%D0%BE%D0%B4)[.](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D0%BD%D1%83%D1%82%D0%B0_%E2%80%94_%D0%9C%D0%BE%D1%80%D1%80%D0%B8%D1%81%D0%B0_%E2%80%94_%D0%9F%D1%80%D0%B0%D1%82%D1%82%D0%B0#cite_note-2)

Если бы кто-то захотел выполнить много поисков подстроки в одной и той же основной строке, было бы целесообразно построить вспомогательный индекс для этой основной строки, чтобы облегчить поиск. Полезная структура индекса, которая может быть построена за линейное время по длине главной строки, и в то же время позволяет выполнять поиск подстроки линейное время равное длине подстроки, была впервые обнаружена Вайнером. Идея алгоритма была в нахождении первых символов суффикса, которые находились в уже построенном дереве. Суффиксы просматривались от самого короткого к самому длинному, а для быстрого поиска использовались по два массива размера алфавита на каждую вершину, что затрудняло как понимание алгоритма, так и его реализацию и эффективность, особенно в плане занимаемой памяти.

Маккрейт в 1976 году предложил свой алгоритм, в котором порядок добавления суффиксов заменен на обратный (то есть от большего к меньшему), а для быстрого вычисления места, откуда нужно продолжить построение нового суффикса, достаточно суффиксной ссылки в каждой вершине.

# Принцип работы алгоритма

## Основная идея алгоритма

1. Последний символ суффикса *S* не может появляться где-либо еще кроме *S*.

Если строка не удовлетворяет условию, ее можно расширить до строки, которая соответствует, дополнив ее новым символом. Например, строка *abab* неприемлема, но ее можно дополнить допустимой строкой *ababc*. Если строка S удовлетворяет условию (пункт 1), то никакой суффикс *S* не является префиксом другого суффикса *S*. Это приводит к существованию конечного узла в *T* для каждого суффикса *S*, поскольку любые два суффикса *S* в итоге идут разными путями в *T*.

Пусть n представляет длину строки *S*. Чтобы позволить алгоритму Маккрейта работать за линейное время, на форму *T* накладываются три ограничения (пункт 2). Вместе они приводят к тому, что дерево *T*, представляющее *S*, является многоходовым деревом Патриции (*Patricia Merkle Tree*) и, таким образом, содержит не более *n* нетерминальных узлов.

1. Три ограничения, которые накладываются на cуффиксное дерево.
   1. Дуга *T* может представлять любую непустую подстроку *S*.
   2. Каждый нетерминальный узел *T*, кроме корня, должен иметь не менее двух дуг-потомков.
   3. Строки, представленные одноуровневыми дугами *T*, должны начинаться с разных символов.

Ограничения в пункте 1, пункте 2.2 и 2.3 гарантируют, что разделенный путь может быть назван однозначно путем объединения строк на его дугах.

Алгоритм Маккрейта начинает с пустого дерева *T* и добавляет пути, соответствующие суффиксам *S*, по одному, от самого длинного до самого короткого. Для примера работы алгоритма Маккрейта разберём строку *ababc*, далее называемую строка *S*, и построим на её основе суффиксное дерево.

Мы определяем *suf* как суффикс *S*, начинающийся с символа в позиции *i*. На каждом шаге алгоритм вставляет путь, соответствующий строке suf, в дерево *Ti-1*, чтобы создать дерево *Ti*. Мы определяем headi как самый длинный префикс *suf*, который также является префиксом *sufj*, для некоторого j < i. Определим *head* как самый длинный префикс *suf*, который существует в дереве *Ti-1*, а tail равное *sufi - headi*. Для строки **ababc**, например, *suf3*= abc, *head3*= ab и *tail* = c.

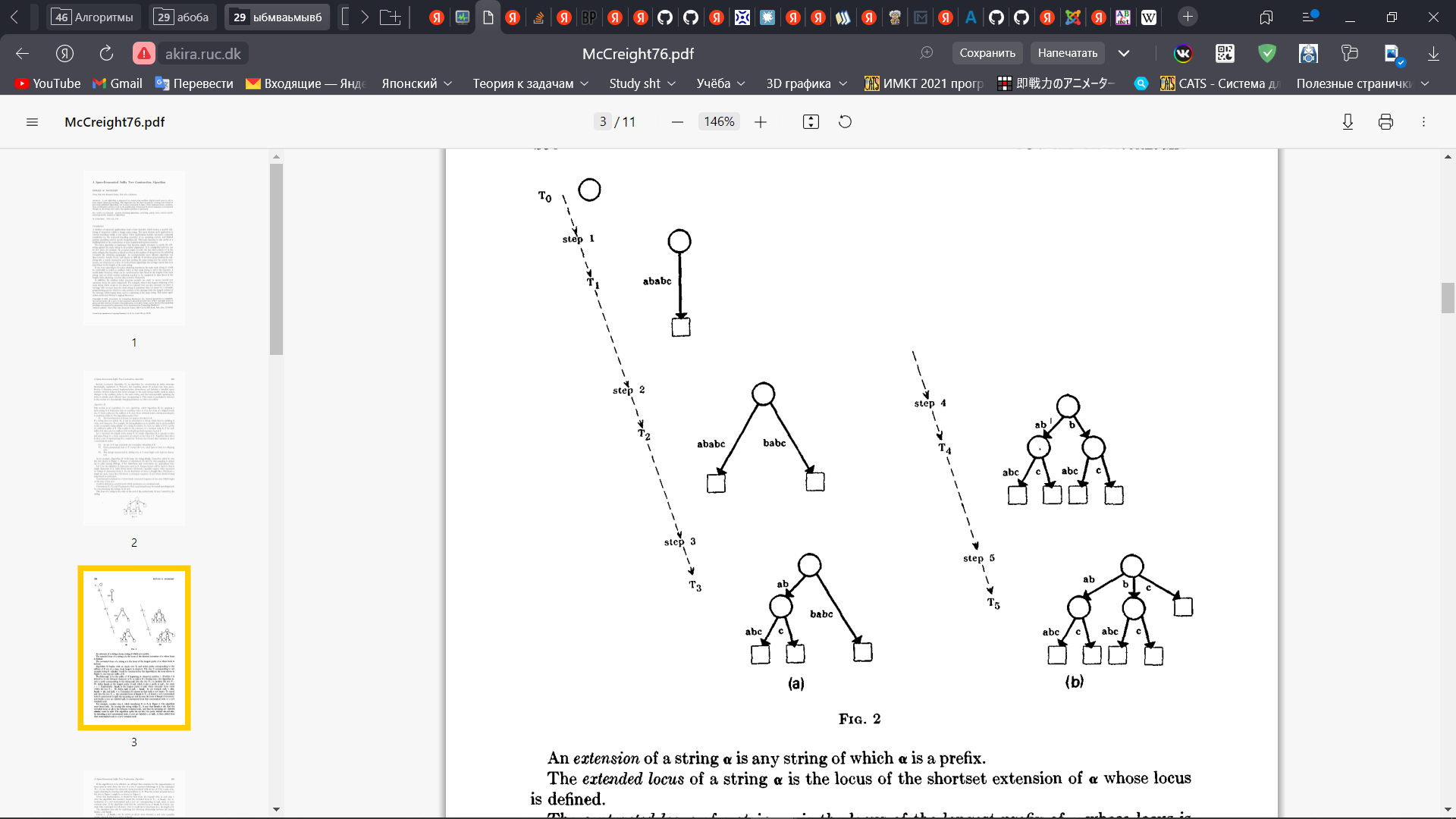


Рисунок 1-Представление построения суффиксного дерева для строки ababc

На шаге 3 (Рисунок 1) мы преобразуем дерево, добавляя в него *suf3*. Прослеживая строку в дереве *T2*, алгоритм видит, что *head3* — это *ab* и, что лист *ab* является крайним левым конечным узлом, и что его входящая дуга (обозначенная *ababc*) должна быть разделена на *head3*= ab и *tail3*= c.

Важным аспектом алгоритма является использование двух разных реализаций функции поиска. Первый, называемый быстрым поиском (*fastfind*), имеет дело с ситуацией, когда мы заранее знаем, что путь поиска полностью содержится в каком-то существующем пути и мы знаем откуда он начинается. Это знание позволяет нам намного быстрее найти искомый узел, используя сжатые ребра дерева в качестве быстрых путей.

Второй же функцией поиска, реализуемой в алгоритме, является функция медленного поиска (*slowfind*), которая следует по пути буква за буквой.

Применение быстрого поиска является основной особенностью алгоритма МакКрейта и играет центральную роль в его производительности (вместе с суффиксными ссылками).

## Схема алгоритма Маккрейта:

*T* := two-node tree with one edge labeled by *p1* = *S*;

**for** *i* := 2 **to** *n* **do begin**

{ insert next suffix *pi* = *S[i. .n]* }

localize *headi* as *head(pi,T)*,

starting the search from *suf* [father(*headi* -1)]

and using *fastfind* whenever possible;

*T* := insert(*pi*,*T*);

**end**

Если выполнять поиск *head* грубым способом, каждый раз начиная от корня, то время выполнения алгоритма будет квадратично. Ключом к оптимизации алгоритма является соотношения *headi*и *headi-1* . Следовательно, при поиске следующего *head* мы можем начать поиск не с корня, а с некоторого узла в глубине дерева, используя суффиксные ссылки.

## Схема алгоритма Маккрейта с использованием суф. ссылок:

*T* := two-node tree with one edge labeled by *p1* = *S*;

**for** *i* := 2 **to** *n* **do begin**

{ insert next suffix *pi* = *S[i. .n]* }

let (*β* be the label of the edge (father[*headi-1*],*headi-1*);

let *γ* be the label of the edge (*headi-1*,*leafi-1*);

*u* := *suf* [father[*head i-1*]];

*v* := *fastfind*(*u*, *β*);

*suf* [*head i-1*] := *v*;

**if** *v* has only one son **then**

{ *v* is a newly inserted node } *head i* := *v*

**else** *headi* := *slowfind*(*v*,*γ*);

create a new leaf *leafi*, make *leafi* a son of *headi*;

label the edge (*headi*, *leafi*) accordingly;

**end**

# Заключение

Первый алгоритм генерации суффиксных деревьев за линейное время был открыт Вайнером. Ясное описание алгоритма Вайнера с дополнительными сведениями содержится в неопубликованных конспектах лекций Кнута. Современное состояние сопоставления с образцом, включая алгоритм Вайнера, хорошо представлено в книге Ахо, Хопкрофта и Ульмана. Пратт, следуя работе Вайнера, разработал неопубликованный алгоритм для решения этой проблемы несколько другим способом. Все эти алгоритмы решают задачу за линейное время.

Разница между алгоритмом Mаккрейта и другими вышеперечисленными алгоритмами заключается в том, что алгоритм Mаккрейта может использовать меньше места для хранения данных. Количество узлов, генерируемых каждым алгоритмом, примерно одинаково, хотя исходный алгоритм Вайнера генерирует немного больше, чем другие. Кроме того, информационное содержание каждого узла примерно одинаково для всех алгоритмов, за одним существенным исключением. Исключением является то, что, обрабатывая методом left-to-right и никогда не расширяя какую-либо подстроку влево, алгоритм Mаккрейта избегает левого указателя на узел для каждого символа алфавита, который требуется для других алгоритмов. Это обеспечивает примерно 25-процентную экономию пространства данных по сравнению с версией алгоритма с хеш-кодом и аналогичными версиями других алгоритмов. Примерно такую же экономию можно было бы ожидать и для других представлений дерева.

Сокращая затраты по времени и памяти, алгоритм Маккрейта строит суффиксное дерево из заданной строки. Данный способ хранения информации существенно упрощает поиск конкретных слов или строк в больших объемах текста. Однако этот алгоритм уступает по качеству алгоритму Уконена, который является онлайн алгоритмом и используется в браузерах, при поисковых запросах.

# Список источников

1. <https://habr.com/ru/post/681940/>
2. https://habr.com/ru/post/258121/
3. <https://slideplayer.com/slide/12993932/>
4. https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм\_МакКрейта
5. Maxime Crochemore, Wojciech Rytter – “Jewels of stringology”
6. <https://www.cs.helsinki.fi/u/tpkarkka/opetus/11s/spa/lecture09.pdf>
7. <https://ru.wikipedia.org/wiki/Суффиксное_дерево>
8. <https://intuit.ru/studies/courses/1116/145/lecture/4012?page=6>
9. C. N. Storm – “McCreight's suffix tree construction algorithm”
10. E. M. McCreight – “A Space-Economical Suffix Tree Construction Algorithm”
11. <https://marknelson.us/posts/1996/08/01/suffix-trees.html>
12. Maxime Crochemore, Wojciech Rytter – “Text algotithms”
13. <https://habr.com/ru/post/198682/>
14. [Dan Gusfield, Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology // Cambridge University Press](http://www.academia.edu/3146231/Algorithms_on_strings_trees_and_sequences_computer_science_and_computational_biology)
15. <http://proteus2001.narod.ru/gen/txt/14/suff-tress.html>
16. <https://ru.wikipedia.org/wiki/Суффиксный_автомат>
17. <https://ppt-online.org/276323>
18. <https://habr.com/ru/post/533774/>
19. <https://users.math-cs.spbu.ru/~okhotin/teaching/tcs2_2019/okhotin_tcs2alg_2019_l3.pdf>
20. <https://www.youtube.com/watch?v=wyFvEECgsWs>
21. <https://www.youtube.com/watch?v=F41tQ8LS4XI>
22. <https://www.youtube.com/watch?v=ri5fE6CeCLs>
23. Mohri M., Moreno P., Weinstein E. [General suffix automaton construction algorithm and space bounds](https://www.sciencedirect.com/science/article/pii/S0304397509002370)
24. [Larsson N. J.](https://www.wikidata.org/wiki/Q102409399) [Extended application of suffix trees to data compression](https://ieeexplore.ieee.org/document/488324)
25. Паращенко Д. А. [Обработка строк на основе суффиксных автоматов](http://is.ifmo.ru/diploma-theses/paraschenko/doc.pdf)
26. [Moritz Maaß Suffix Trees and their Applications.](https://web.archive.org/web/20070928011830/http:/www.informatics.ru/library/27/suffix-trees.rar)
27. <https://translated.turbopages.org/proxy_u/en-ru.ru.ca5ba430-639576ec-e61d83fc-74722d776562/https/en.wikipedia.org/wiki/Edward_M._McCreight>
28. <https://ethereum.org/en/developers/docs/data-structures-and-encoding/patricia-merkle-trie/>