Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Д. А. Паращенко

Обработка строк на основе суффиксных автоматов

Бакалаврская работа

Научный руководитель: А. С. Станкевич

Санкт-Петербург
2007

Оглавление

Введение **3**

Глава 1. Основные понятия **4**

1.1. Строки 4
1.2. Правые контексты 5
1.3. Суффикс функция 7
1.4. Суффиксный бор 7
1.5. Суффиксное дерево 9
1.6. Суффиксный автомат 10
1.7. Сжатый суффиксный автомат 12

Глава 2. Построение суффиксного дерева по суффиксному автомату **15**

2.1. Связь суффиксного автомата и суффиксного бора 15
2.2. Связь сжатого суффиксного автомата и суффиксного дерева 16
2.3. Построение суффиксного бора по суффиксному автомату . . 17
2.4. Построение суффиксного дерева по суффиксному автомату 19
2.5. Алгоритм построения суффиксного дерева за линейное время 22

Глава 3. Применение суффиксного автомата для решения строковых задач **25**

3.1. Реализация суффиксного дерева на основе суффиксного автомата 25
3.1.1. Корневая вершина 26
3.1.2. Конечные вершины 27
3.1.3. Суффиксная ссылка 27
3.1.4. Дуги суффиксного дерева 27
3.2. Задача о наибольшем общем префиксе двух суффиксов . . . 28

Заключение **34**

Список литературы **35**

Введение

В настоящее время для решения большого числа строковых задач применяются суффиксные деревья [1]. При этом все известные алгоритмы построения суффиксного дерева за линейное время [1] достаточно сложны для понимания и реализации.

В настоящей работе разработан достаточно простой алгоритм построения суффиксного дерева за линейное время, содержащий в качестве одного из своих этапов построение суффиксного автомата [1]. Таким образом, помимо алгоритмов Вайнера [2], Мак-Крейта [3] и Укконена [4] предложен еще один алгоритм построения суффиксного дерева [1] за линейное время. Кроме этого в работе проведено сравнение времени их работы, а также сложности их реализации.

Поясним как появилась идея использовать суффиксный автомат для решения рассматриваемой задачи. Ввиду того, что суффиксные деревья и суффиксные автоматы являются родственными структурами данных (суффиксный автомат является минимизированным суффиксным бором [5], а суффиксное дерево - сжатым суффиксным бором [5]), то авторы посчитали целесообразным для решения строковых задач вместо суффиксных деревьев использовать суффиксные автоматы. Одним из преимуществ этого подхода является тот факт, что суффиксный автомат является более простой структурой данных, а алгоритм его построения значительно проще в реализации, чем алгоритм построения суффиксного дерева.

В настоящей работе предложен также метод, позволяющий во многих алгоритмах вместо суффиксного дерева использовать суффиксный автомат. Суть этого метода состоит в изменении алгоритма таким образом, чтобы в нем вместо операций над суффиксным деревом использовались операции над суффиксным автоматом.

Глава 1.

Основные понятия

1.1. Строки

Определение 1. [5, стр. 3] Пусть Σ — конечное множество, называемое *алфавитом*. Под Σ^* будем понимать *множество слов* над алфавитом Σ. *Пустое слово* обозначим за ε. Множество всех непустых слов обозначим за $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

Определение 2. [5, стр. 3] Для слова w, под $|w|$ будем понимать *длину* w, то есть число символов в w.

Определение 3. [5, стр. 4] Строка w называется *подстрокой* строки u, если существуют такие слова x и y, что $u = xwy$.

Определение 4. [5, стр. 4] Строка w называется *префиксом* строки u, если существует такое слово y, что $u = wy$.

Определение 5. [5, стр. 4] Строка w называется *суффиксом* строки u, если существует такое слово x, что $u = xw$.

Определение 6. Множество всех подстрок слова x обозначим за $Fact(x)$.

Определение 7. Множество всех суффиксов слова x обозначим за $Suff(x)$.

Определение 8. [5, стр. 4] Префикс длины k слова w обозначим за $w[1 \dots k]$.

Определение 9. k-ый символ слова w обозначим за $x[k]$.

1.2. Правые контексты

Определение 10. [5, стр. 116] *Правым контекстом* строки x в строке y называется $R_y(x) = x^{-1}Suff(y) = \{s \in \Sigma^* \mid xs \in Suff(y)\}$.

Неформально говоря, правый контекст строки x в строке y состоит из всех суффиксов строки y, которые расположены непосредственно за строкой x.

Определение 11. [5, стр. 116] Строки u и v называются *конгруэнтными* в строке y ($u \equiv_y v$), если $R_y(u) = R_y(v)$.

Определение 12. [5, стр. 116] Для каждой подстроки u строки y правую позицию первого вхождения u в y обозначим за $end - pos_y(u)$.

Лемма 1. [5, лемма 2.3.1] Пусть $u, v \in Fact(y)$ и $|u| \leq |v|$. Тогда:

- если u является суффиксом v, то $R_y(v) \subseteq R_y(u)$;
- если $R_y(v) = R_y(u)$, то $end - pos_y(u) = end - pos_y(v)$ и $u \in Suff(v)$.

Лемма 2. [5, лемма 2.3.2] Пусть $u, v, w \in Fact(y)$. Если u является суффиксом v, а v — суффиксом w, и $u \equiv_y w$, тогда $u \equiv_y v \equiv_y w$.

Лемма 3. [5, лемма 2.3.3] Пусть $u, v \in \Sigma^*$. Тогда правые контексты u и v либо сравнимы по включению, либо не пересекаются — выполнено хотя бы одно из следующих утверждений:

- $R_y(u) \subseteq R_y(v)$;
- $R_y(v) \subseteq R_y(u)$;
- $R_y(u) \cap R_y(v) = \varnothing$.

Поскольку отношение конгруэнтности является отношением эквивалентности, то все строки разбиваются на классы эквивалентности (конгруэнтности).

Определение 13. *Множеством представителей* правого контекста C в строке y называется множество $Repr_y(C) = \{x \in Fact(y) \mid R_y(x) = C\}$.

Теперь приведем определения наибольшего и наименьшего представителя правого контекста.

Определение 14. *Наибольшим представителем* правого контекста C в строке y называется элемент множества $Repr_y(C)$, имеющий наибольшую длину.

Определение 15. *Наименьшим представителем* правого контекста C в строке y называется элемент множества $Repr_y(C)$, имеющий наименьшую длину.

Лемма 4. Множество представителей правого контекста состоит из суффиксов наибольшего представителя, длина которых не меньше длины наименьшего представителя этого контекста. Более формально:

Пусть w_{max} и w_{min} — наибольший и наименьший представители правого контекста C в строке y. В таком случае, s принадлежит $Repr_y(C)$ тогда и только тогда, когда s является суффиксом w_{max} и $|s| \geq |w_{min}|$.

$\triangleright$ Пусть $s \in Repr_y(C)$. Покажем, что $s \in Suff(w_{max})$ и $|s| \geq |w_{min}|$.

По определению 14 $w_{max} \in Repr_y(C)$ и $|w_{max}| \geq |s|$. По лемме 1 из этого следует, что $s \in Suff(w_{max})$. С другой стороны, из определения 15 следует $|s| \geq |w_{min}|$, что и требовалось доказать.

Пусть $s \in Suff(w_{max})$ и $|s| \geq |w_{min}|$. Покажем, что $s \in Repr_y(C)$.

По определению 15 и лемме 1 $w_{min} \in Suff(s)$. Из определений 14 и 15 следует, что $w_{max} \equiv_y w_{min}$. Тогда, по лемме 2 $w_{max} \equiv_y s \equiv_y w_{min}$, что и требовалось доказать. $\triangleleft$

Ниже приведено важное следствие этой леммы.

Следствие 5. Множество представителей некоторого правого контекста однозначно задается его наибольшим представителем и длиной его наименьшего представителя.

$\triangleright$ Рассмотрим правый контекст $R_y(x)$. Пусть w_{max} — его наибольший представитель, и len_{min} — длина его наименьшего представителя. Тогда, по теореме 4 $Repr_y(R_y(x)) = \{z \mid z \in Suff(w_{max}) \wedge |z| \geq len_{min}\}$. $\triangleleft$

Следствие 6. Для любого правого контекста C и числа len существует не более одного слова y длины len, такого, что $R_w(y) = C$.

$\triangleright$ Предположим, что существуют строки y_1 и y_2 длины len такие, что $C = R_w(y_1) = R_w(y_2)$. Докажем, что $y_1 = y_2$. По лемме 1 либо $y_1 \in Suff(y_2)$, либо $y_2 \in Suff(y_1)$. Из того, что y_1 и y_2 являются суффиксами друг друга, и их длины равны, следует, что $y_1 = y_2$, что и требовалось доказать. $\triangleleft$

1.3. Суффикс функция

Определение 16. [5, стр. 118] Рассмотрим на множестве $Fact(w)$ функцию s_w, называемую *суффикс функцией* строки w, которая определена для всех $x \in Fact(w) \setminus \{\varepsilon\}$ и равна наидлиннейшему неконгруэнтному x суффиксу x.

Теорема 7. Пусть $x_2 = s_w(x_1)$. Тогда длина наибольшего представителя $R_w(x_2)$ на единицу меньше длины наименьшего представителя $R_w(x_1)$.

$\triangleright$ Из определения 16 следует, что x_2 является суффиксом строки x_1 и $|x_2| < |x_1|$.

Рассмотрим строку y, являющуюся суффиксом строки x_1, длина которого на единицу больше длины строки x_2. То есть $y = x_1[|x_1| - |x_2| - 1 \ldots |x_1|]$.

Из определения 16 следует, что $y \equiv_w x_1$ и x_2 является наибольшим представителем $R_w(x_2)$. Из леммы 2 и того факта, что $y \not\equiv_w x_1$, следует, что y является наименьшим представителем $R_w(y) = R_w(x_1)$. По построению $|y| = 1 + |x_2|$, что и требовалось доказать. $\triangleleft$

Лемма 8. Для любой непустой строки x из $Fact(w)$ имеет место соотношение $R_w(x) \subset R_w(s_w(x))$.

$\triangleright$ По определению 16 $s_w(x)$ является суффиксом x, $|s_w(x)| < |x|$ и $R_w(x) \neq R_w(s_w(x))$.

Для завершения доказательства достаточно воспользоваться леммой 1. $\triangleleft$

1.4. Суффиксный бор

Определение 17. *Суффиксным бором* T строки w называется минимальное ориентированное дерево с корнем. Каждая дуга помечена некоторым символом из Σ (дуговой меткой). Никакие две дуги, выходящие из одной вершины, не могут иметь одинаковых пометок. Ровно $|w|+1$ вершин имеют пометку, означающую, что вершина является конечной. Главная особенность суффиксного бора состоит в том, что для каждой помеченной (конечной) вершины конкатенация меток на пути от корня к данной вершине в точности составляет (произносит) некоторый суффикс строки w.

Определение 18. [1, стр. 122] *Метка пути* от корня до некоторой вершины — это конкатенация символов, помечающих дуги пути в порядке их прохождения. *Путевая метка вершины* — это метка пути от корня до этой вершины.

Определение 19. [1, стр. 122] Для любой вершины v *строковой глубиной* v называется число символов в путевой метке v.

Определение 20. *Степенью вершины* v будем называть число исходящих из нее дуг.

Определение 21. Для каждой вершины u суффиксного бора под множеством $R_T(u)$ будем понимать множество строк, являющихся конкатенациями меток на путях от вершины u к помеченным вершинам, находящимся в поддереве вершины u.

Определение 22. Будем говорить, что вершине v суффиксного бора соответствует правый контекст $R_w(x)$, если $R_T(v) = R_w(x)$.

Определение 23. Вершины v и u называются *эквивалентными*, если $R_T(v) = R_T(u)$.

На рис. 1 приведен суффиксный бор строки *aabb*. Эквивалентные вершины выделены прямоугольниками.

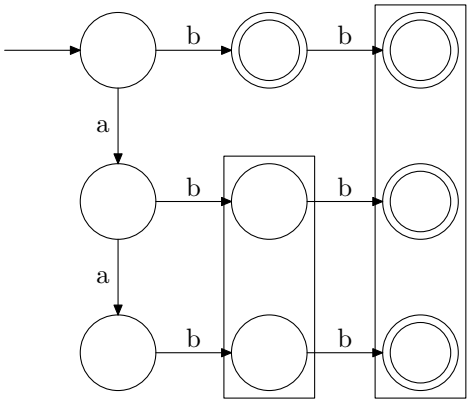

Рис. 1. Суффиксный бор строки *aabb*

Теорема 9. Каждой вершине v суффиксного бора соответствует правый контекст путевой метки v.

$\triangleright$ Обозначим путевую метку v за x. Требуется доказать, что $R_T(v) = R_w(x)$.

Покажем, что $y \in R_T(v)$ тогда и только тогда, когда $y \in R_w(x)$. $y \in R_T(v)$ означает, что xy является суффиксом w, что по определению 10 означает, что $y \in R_w(x)$. $\triangleleft$

Теорема 10. Каждому непустому правому контексту в w соответствует, по крайней мере, одна вершина суффиксного бора.

$\triangleright$ Пусть $R_w(x)$ — непустой правый контекст. Так как $R_w(x)$ непуст, x является подстрокой w. Из того, что x является подстрокой w, следует, что в суффиксном боре содержится вершина v с путевой меткой равной x.

Для завершения доказательства осталось воспользоваться теоремой 9. $\triangleleft$

1.5. Суффиксное дерево

Имеется несколько определений (сжатого) суффиксного дерева ([5, стр. 108], [1, стр. 121]). В настоящей работе под суффиксным деревом понимается следующая структура данных.

Определение 24. [5, стр. 108] *Сжатым суффиксным деревом* строки w, для простоты называемым *суффиксным деревом*, называется структура, получающаяся из суффиксного бора в результате удаления всех вершин степени один, которые не являются конечными.

На рис. 2 приведено суффиксное дерево строки *aabb*. Эквивалентные вершины выделены прямоугольниками.

Определение 25. [1, стр. 122] Путь, который кончается в середине дуги (u, v), делит метку (u, v) в своей точке назначения. Определим метку этого пути как путевую метку u с добавлением символов дуги (u, v) до точки назначения, которая делит дугу.

Замечание. Теорема 10 не применима к суффиксному дереву, так как непустому правому контексту в w может соответствовать любое число вершин суффиксного дерева (в том числе и ноль).

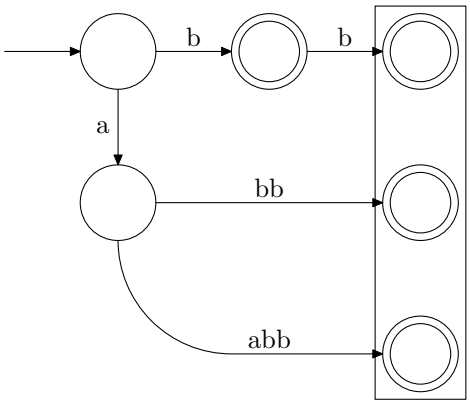

Рис. 2. Суффиксное дерево строки *aabb*

1.6. Суффиксный автомат

Определение 26. [5, стр. 121] *Суффиксным автоматом* A строки w называется наименьший детерминированный конечный автомат над алфавитом Σ, допускающий все суффиксы строки w и только их.

На рис. 3 приведен суффиксный автомат строки *aabb*.

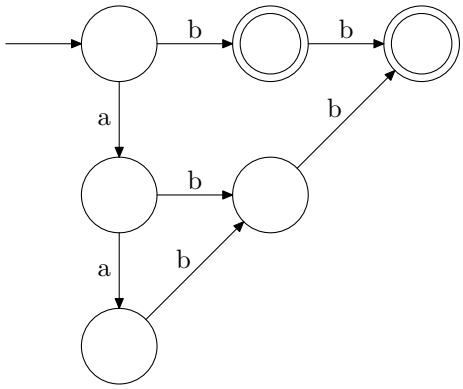

Рис. 3. Суффиксный автомат строки *aabb*

Определение 27. Будем говорить, что состоянию s суффиксного автомата соответствует правый контекст $R_w(x)$, если $R_A(s) = R_w(x)$.

Предложение 11. [5] Существует взаимно однозначное соответствие между множеством состояний суффиксного автомата и множеством непустых правых контекстов в w.

Поэтому, можно говорить о правом контексте состояния, имея ввиду соответствующий ему правый контекст, и о представителях состояния, имея ввиду представителя его правого контекста.

Предложение 12. Множество всех строк, которые переводят автомат из начального состояния в некоторое состояние s, совпадает с множеством представителей правого контекста этого состояния.

$\triangleright$ Для доказательства этого факта достаточно заметить, что суффиксный бор является детерминированным конечным автоматом, и воспользоваться определениями 26, 13 и теоремой 9. $\triangleleft$

Определение 28. Пусть s — состояние, в котором окажется суффиксный автомат после получения на вход непустого слова x. *Суффиксной ссылкой* из состояния s называется ссылка, ведущая из состояния s в состояние, в котором окажется автомат после получения на вход слова $s(x)$. Обозначим это состояние за $suffix(s)$.

Определение 29. Длину наибольшего представителя состояния s обозначим за $repr_{max}(s)$.

Замечание. В связи с тем, что на сегодняшний день известен алгоритм построения суффиксного автомата за линейное время ([5, стр. 126]) (который в процессе построения суффиксного автомата считает суффикс-функцию и длину наибольшего представителя каждого состояния), можно считать, что суффиксный автомат, как структура данных, имеет суффиксные ссылки и для каждого состояния известна длина его наибольшего представителя.

Предложение 13. Пусть состояние s суффиксного автомата не является начальным. Тогда длина наименьшего представителя состояния s равна $1 + repr_{max}(suffix(s))$.

$\triangleright$ Требуемое равенство следует из определений 28 и 29, а также из теоремы 7. $\triangleleft$

Теорема 14. Пусть непустое слово x является некоторым представителем состояния s_1 суффиксного автомата. Пусть s_2 — состояние, в которое придет суффиксный автомат, приняв на вход слово $x[2 \dots |x|]$.

Тогда:

- если $|x| = repr_{max}(suffix(s_1)) + 1$, то s_2 совпадает с $suffix(s_1)$

- если $|x| > repr_{max}(suffix(s_1)) + 1$, то s_2 совпадает с s_1

$\triangleright$ Предположим, что $|x| > repr_{max}(suffix(s_1))+1$. Из определений 28 и 16 следует, что слово $x[2 \dots |x|]$ является представителем $R_w(x)$. Это означает, что суффиксный автомат, приняв на вход слово $x[2 \dots |x|]$, окажется в состоянии s_1.

Предположим, что $|x| = repr_{max}(suffix(s_1))+1$. В этом случае, слово $x[2 \dots |x|]$ является представителем $R_w(s_w(x))$, что по определению 16 означает, что суффиксный автомат, приняв на вход слово $x[2 \dots |x|]$, окажется в состоянии $suffix(s_1)$. $\triangleleft$

Определение 30. [5, стр. 132] Будем говорить, что состояние s суффиксного автомата является *вилкой*, если выполнено хотя бы одно из следующих условий:

- состояние s является допускающим
- из состояния s исходит более одного перехода

1.7. Сжатый суффиксный автомат

Определение 31. [5, стр. 132] *Сжатым суффиксным автоматом* A строки w называется автомат над алфавитом Σ^+, который получается из суффиксного автомата в результате удаления всех состояний, не являющихся вилками.

На рис. 4 приведен сжатый суффиксный автомат строки *aabb*.

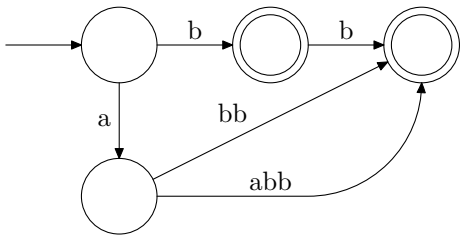

Рис. 4. Сжатый суффиксный автомат строки *aabb*

Теорема 15. Рассмотрим сжатый суффиксный автомат A строки w. Пусть в A имеется переход по слову x из состояния s_1 в состояние s_2. Тогда $x = w[|w| - |y| - |x| + 1 \dots |w| - |y|]$, где y — любая строка из $R_A(s_2)$.

$\triangleright$ Из того, что в автомате A имеется переход по слову x из состояния s_1 в состояние s_2 и $y \in R_A(s_2)$, следует, что $xy \in R_A(s_1)$. Это, в свою очередь, означает, что $xy \in Suff(w)$.

Поскольку слово xy является суффиксом w, имеет место равенство $xy = w[|w| - |xy| + 1 \dots |w|]$.

Отсюда следует, что $x = w[|w| - |xy| + 1 \dots |w| - |y|] = w[|w| - |x| + |y| + 1 \dots |w| - |y|]$, что и требовалось доказать. $\triangleleft$

Определение 32. Введем функцию $longest(s) = \max_{x \in R_A(s)} |x|$, равную наибольшей длине слова, допускаемого суффиксным автоматом, стартующим из состояния s.

Определение 33. Для сжатия суффиксного автомата введем функцию $fork$, заданную следующим образом:

- если состояние s является вилкой, то $fork(s) = s$
- если состояние s не является вилкой, то $fork(s) = t$, где t — состояние, в которое ведет единственный переход из состояния s

Следствие 16. Рассмотрим суффиксный автомат строки w. Пусть имеется переход по символу c из состояния s_1 в состояние s_2. Тогда:

- В сжатом суффиксном автомате строки w имеется переход из состояния s_1 в состояние $fork(s_2)$ по некоторому слову y, первый символ которого равен c.
- При этом $y = w[|w| - longest(s_2) \dots |w| - longest(fork(s_2))]$.

$\triangleright$ Заметим, что все состояния суффиксного автомата на пути от s_2 до $fork(s_2)$, кроме последнего, не являются вилками и, следовательно, отсутствуют в сжатом суффиксном автомате.

Существование указанного в условии перехода следует из определения 31.

Для доказательства второго факта достаточно заметить, что $1 + longest(s_2) = |y| + longest(fork(s_2))$, и воспользоваться теоремой 15. $\triangleleft$

Таким образом, для сжатия суффиксного автомата строки w достаточно вычислить функции $fork$ и $longest$, что можно сделать за время $O(|w|)$.

Глава 2.

Построение суффиксного дерева по суффиксному автомату

2.1. Связь суффиксного автомата и суффиксного бора

Определение 34. Будем говорить, что вершина u суффиксного бора *эквивалентна* состоянию s суффиксного автомата, если $R_T(u) = R_A(s)$.

Такое определение вполне логично. Действительно, суффиксный бор можно рассматривать как детерминированный конечный автомат над алфавитом Σ и считать состояния двух автоматов эквивалентными, если автоматы стартуя из этих состояний допускают одинаковые множества строк.

Теорема 17. Для каждой вершины суффиксного бора строки w существует единственное эквивалентное ей состояние суффиксного автомата строки w.

$\triangleright$ Для доказательства этой теоремы достаточно рассматривать суффиксный бор как детерминированный конечный автомат над алфавитом Σ и воспользоваться определением 26. $\triangleleft$

Теперь попробуем по некоторому состоянию s суффиксного автомата найти вершину суффиксного бора, эквивалентную ему.

Для этого зафиксируем некоторый представитель правого контекста этого состояния. При этом вершина суффиксного бора с такой путевой меткой будет искомой.

Найденная вершина зависит от того, какой представитель правого контекста выбран.

По следствию 6, представитель правого контекста однозначно задается своей длиной. Таким образом, если зафиксировать длину представителя,

то вершина суффиксного бора, эквивалентная состоянию s, будет найдена однозначно.

С другой стороны, представитель состояния s — это путевая метка найденной вершины (по построению).

Из изложенного следует утверждение.

Лемма 18. Существует биекция между множеством вершин суффиксного бора и множеством пар $\langle s, |r| \rangle$, где s — состояние суффиксного автомата, а r — некоторый представитель s.

2.2. Связь сжатого суффиксного автомата и суффиксного дерева

Теорема 19. [5, стр. 132] Сжатый суффиксный автомат является минимизированным суффиксным деревом.

Лемма 20. [5, стр. 132] Для каждой вершины суффиксного дерева строки w существует единственное эквивалентное ей состояние сжатого суффиксного автомата строки w.

Лемма 21. Каждой дуге суффиксного дерева соответствует некоторый переход сжатого суффиксного автомата,а каждому переходу сжатого суффиксного автомата — хотя бы одна дуга суффиксного дерева.

$\triangleright$ Утверждение этой теоремы следует из теоремы 19. $\triangleleft$

Замечание. Как и в случае с суффиксным бором и суффиксным автоматом, некоторому состоянию сжатого суффиксного автомата может соответствовать несколько вершин суффиксного дерева.

Повторив рассуждения, аналогичные рассуждениям из предыдущего раздела , можно показать, что существует биекция между множеством вершин суффиксного дерева и множеством пар $\langle s, |r| \rangle$, где s — состояние сжатого суффиксного автомата, а r — некоторый представитель s.

2.3. Построение суффиксного бора по суффиксному автомату

Суффиксный бор можно построить в процессе рекурсивного обхода суффиксного автомата.

Ниже приведен алгоритм построения суффиксного бора по суффиксному автомату.

```
1     public SuffixTrie buildTrie(SuffixAutomaton auto)
          {
2          SuffixTrie = new SuffixTrie();
3          int root = walk(trie, auto, auto.
             getStartState());
4          trie.setRootNode(root);
5          return trie;
6     }
7
8     private int walk(SuffixTrie trie,
9                      SuffixAutomaton auto,
10                     int state)
11    {
12         int node = trie.createNode();
13         if (auto.isFinal(state)) {
14             trie.setLeaf(node, true);
15         }
16         for (Edge edge: auto.getEdges(state)) {
17             int newState = fork(edge.getTarget());
18             int newNode = walk(trie, auto, newState);
19             trie.addEdge(node, newNode, edge.getChar
                 ());
```

```
20          }
21          return node;
22      }
```

Докажем, что buildTrie(A) строит суффиксный бор по суффиксному автомату A. Для этого достаточно доказать, что walk(T, A, s) добавляет в бор T вершину, эквивалентную состоянию s суффиксного автомата A, и возвращает ее.

Будем доказывать методом математической индукции по длине строки x, что $x \in R_A(s)$ тогда и только тогда, когда x принадлежит поддереву возвращаемой методом walk вершины.

$\triangleright$ База индукции.

Пусть $x = \varepsilon$. $\varepsilon \in R_A(s)$ тогда и только тогда, когда состояние s является конечным. Из строк 13-15 видно, что вершина $node$ будет отмечена конечной тогда и только тогда, когда $\varepsilon \in R_A(s)$.

Переход индукции.

Рассмотрим случай, когда $x \neq \varepsilon$. $R_A(s)$ по определению является множеством всех строк, допускаемых автоматом A, стартующим из состояния s.

Покажем, что если $x \in R_A(s)$, то бор поддерево возвращаемой методом вершины бора T будет содержать строку x.

$x \in R_A(s)$ означает, что автомат A, стартующий из состояния s, допускает строку x. Это, в свою очередь, означает, что в автомате A существует переход из состояния s по символу $x[1]$. Следовательно, этот переход будет обработан в цикле на сроках 16-20. По предположению индукции, поддерево вершины $newNode$ содержит строку $x[2 \dots |x|]$. Отсюда следует, что поддерево возвращаемой методом вершины будет содержать строку x.

Теперь докажем, что если $x \notin R_A(s)$, то поддерево возвращаемой методом вершины бора T не будет содержать строку x. Будем доказывать это утверждение от противного. Предположим, что указанное поддерево содержит строку x. Это означает, что есть переход из вершины $node$ по символу $x[1]$. Переход из корневой вершины мог добавиться только в строке 19. Отсюда следует, что поддерево вершины $newNode$ должно было содержать строку $x[2 \dots |x|]$, что по предположению индукции означает, что автомат A должен был принимать строку $x[2 \dots |x|]$, стартуя из состояния s_2, а следовательно, и строку x, стартуя из состояния s, что противоречит условию

$x \notin R_A(s)$. Следовательно, предположение о том, что поддерево вершины *node* содержит строку x неверно. $\lhd$

Замечание. Можно показать, что приведенный выше алгоритм строит суффиксный бор за $O(n)$, где n — число вершин в получившемся боре.

2.4. Построение суффиксного дерева по суффиксному автомату

Суффиксное дерево можно построить в процессе рекурсивного обхода сжатого суффиксного автомата.

Ниже приведен алгоритм построения суффиксного дерева по сжатому суффиксному автомату.

```
1     public SuffixTree buildTree(
        CompactSuffixAutomaton auto) {
2         SuffixTree = new SuffixTree();
3         int root = walk(tree, auto, auto.
            getStartState());
4         tree.setRootNode(root);
5         return tree;
6     }
7
8     private int walk(SuffixTree tree,
9                      CompactSuffixAutomaton auto,
10                     int state)
11    {
12        int node = tree.createNode();
13        if (auto.isFinal(state)) {
14            tree.setLeaf(node, true);
15        }
```

```
16          for (Edge edge: auto.getEdges(state)) {
17              int newNode = walk(tree, auto, edge.
                  getTarget());
18              tree.addEdge(node, newNode,
19                           edge.getBegin(),
20                           edge.getEnd());
21          }
22          return node;
23      }
```

Повторив рассуждения предыдущей главы , получим, что построенное дерево содержит все строки, допускаемые автоматом, и только их. Однако, оно может не оказаться минимальным по числу вершин. Для того, чтобы дерево было минимальным, необходимо, чтобы каждая некорневая вершина была либо конечной, либо содержала более одной исходящей дуги.

Докажем следующее утверждение.

Предложение 22. Если из вершины дерева, возвращаемой методом walk, выходят не более одной дуги, то эта вершина является конечной.

▷ Из того, что из вершины *node*, возвращенной методом walk, выходит не более одной дуги, следует, что в сжатом суффиксном автомате A из состояния *state* выходит не более одного перехода. Это возможно лишь при условии, когда состояние является допускающим. Изложенное, в свою очередь, означает, что при выполнении строк 13-15 вершина *node* была помечен конечной, что и требовалось доказать. ◁

Замечание. Приведенный выше алгоритм строит суффиксное дерево за $O(n)$, где n — число вершин в получившемся дереве.

Учитывая изложенное, для того, чтобы построить суффиксное дерево по суффиксному автомату достаточно научиться строить по суффиксному автомату сжатый суффиксный автомат.

При этом обратим внимание на то, что для того чтобы дерево, построенное по автомату, занимало разумную память, необходимо хранить строки на дугах дерева и переходах автомата не в явном виде, а лишь начальную и конечную позиции их вхождений в строку w.

Эта проблема решается применением теоремы 15, позволяющей вычислять позицию вхождения в строку w слова на переходе сжатого суффиксного автомата.

Для рекурсивного обхода сжатого суффиксного автомата в алгоритме построения суффиксного дерева не требуется строить сжатый суффиксный автомат в явном виде. Вместо этого можно обойтись обычным суффиксным автоматом и функциями $fork$ и $longest$. Этот прием подробно описан в главе 2.

Ниже приведен алгоритм построения суффиксного дерева по суффиксному автомату.

```
1     public SuffixTree buildTree(SuffixAutomaton auto)
          {
2          SuffixTree = new SuffixTree();
3          int root = walk(tree, auto, auto.
             getStartState());
4          tree.setRootNode(root);
5          return tree;
6     }
7
8     private int walk(SuffixTree tree,
9                      SuffixAutomaton auto,
10                     int state)
11    {
12         int node = tree.createNode();
13         if (auto.isFinal(state)) {
14             tree.setLeaf(node, true);
15         }
16         for (Edge edge: auto.getEdges(state)) {
17             int newState = fork(edge.getTarget());
```

```
18              int b = length() - longest(edge.getTarget
                    ()) - 1;
19              int e = length() - longest(newState);
20              int newNode = walk(tree, auto, newState);
21              tree.addEdge(node, newNode, b, e);
22          }
23          return node;
24      }
```

Корректность этого алгоритма следует из корректности предыдущего алгоритма, а также теорем и следствий про суффиксный автомат.

2.5. Алгоритм построения суффиксного дерева за линейное время

Учитывая изложенное в предыдущем параграфе, суффиксное дерево для строки w можно построить следующим образом:

- построить суффиксный автомат для строки w за время $O(|w|)$;
- сжать суффиксный автомат, вычислив для каждого состояния s функции $fork(s)$ и $longest(s)$;
- рекурсивно обойти полученный сжатый суффиксный автомат, как показано в предыдущем параграфе.

С другой стороны, для построения суффиксного дерева для строки w по алгоритму Укконена требуется:

- аккуратно реализовать процесс *продолжения суффикса*;
- применить множество приемов, в совокупности позволяющих сделать время работы алгоритма линейным.

Из изложенного можно сделать вывод, что алгоритм Укконена намного сложнее и в нем больше мест, в которых можно допустить ошибки.

Однако, несмотря на простоту реализации, алгоритм построения суффиксного дерева по суффиксному автомату использует дополнительные $O(|w|)$ памяти для хранения суффиксного автомата и функций $fork$ и $longest$, что, безусловно, является его недостатком.

Из табл. 1 видно, что алгоритм Укконена построения суффиксного дерева всегда работает чуть быстрее приведенного выше алгоритма. Однако разница времен работы алгоритмов не на столько велика, чтобы следовало отказаться от использования алгоритма построения суффиксного дерева по суффиксному автомату.

Таблица 1. Среднее время работы алгоритмов построения суффиксного дерева для строк указанной длины над алфавитом {'a', 'b'}

Длина строки	**Разработанный алг.**	**Алг. Укконена**
100000	0.177	0.125
200000	0.388	0.266
300000	0.596	0.428
400000	0.819	0.580
500000	1.035	0.730
600000	1.264	0.884
700000	1.470	1.047
800000	1.694	1.216
900000	1.923	1.342
1000000	2.153	1.502

Из изложенного можно сделать вывод.

Алгоритм построения суффиксного дерева по суффиксному автомату уступает алгоритму Укконена как по времени работы, так и по памя-

ти. Поэтому в задачах, где константы в скорости работы или количество используемой алгоритмом памяти чрезвычайно важны, предпочтительнее алгоритм Укконена.

Тем не менее, благодаря тому, что алгоритм построения суффиксного дерева по суффиксному автомату реализуется намного быстрее и проще известных алгоритмов построения суффиксного дерева за линейное время, в большинстве случаев его использование является разумным.

Глава 3.

Применение суффиксного автомата для решения строковых задач

Многие алгоритмы решения строковых задач используют суффиксные деревья. В этой главе описывается предложенный автором метод, позволяющий избежать использования суффиксных деревьев. При этом, вместо таких деревьев предлагается применять суффиксные автоматы.

3.1. Реализация суффиксного дерева на основе суффиксного автомата

Заметим, что от суффиксного дерева может потребоваться:

- получать корневую вершину дерева;
- определять, является ли некоторая вершина конечной;
- получать суффиксную ссылку из данной вершины;
- получать список исходящих из данной вершины дуг;
- находить исходящую из данной вершины дугу с заданным первым символом дуговой метки.

Все эти операции можно реализовать на сжатом суффиксном автомате.

В предыдущей главе было выяснено, что существует биекция между вершинами суффиксного дерева и парами $\langle$ состояние сжатого суффиксного автомата, длина представителя этого состояния $\rangle$. Для эффективного использования этого факта необходимо ввести такую нумерацию вершин суффиксного дерева, которая обеспечила бы возможность быстро преобразовывать вершину суффиксного дерева в пару $\langle$ состояние суффиксного автомата, длина представителя этого состояния $\rangle$, а эту пару — в вершину.

Первое преобразование легко сделать, используя $O(|w|)$ дополнительной памяти. Для этого достаточно завести линейный массив, содержащий пары $\langle$ эквивалентное вершине состояние, строковая глубина вершины$\rangle$.

Осталось научиться по состоянию сжатого суффиксного автомата и длине его путевой метки получать номер соответствующей вершины суффиксного дерева. Для этого введем следующую нумерацию вершин суффиксного дерева.

Будем считать, что состояния сжатого суффиксного автомата как-то занумерованы. Введем следующий линейный порядок на вершинах суффиксного дерева. Номер вершины v_1 меньше номера вершины v_2 тогда и только тогда, когда выполнено одно из следующих условий:

- вершинам v_1 и v_2 соответствуют состояния s_1 и s_2 соответственно, причем номер s_1 меньше номера s_2;
- вершинам v_1 и v_2 соответствует одно состояние, и строковая глубина вершины v_1 меньше строковой глубины вершины v_2.

При такой нумерации вершин суффиксного дерева можно достаточно легко по паре $\langle$ состояние, длина представителя $\rangle$ получать номер вершины.

Действительно, если для каждого состояния сжатого суффиксного автомата посчитать наименьший номер num вершины, эквивалентной ему, а также длину len_{min} наименьшего представителя, то номер эквивалентной этому состоянию вершины со строковой глубиной len можно вычислить по формуле $num + len - len_{min}$.

Заметим, что после инициализации за время $O(|w|)$ всех используемых массивов можно делать оба эти преобразования за время $O(1)$.

Рассмотрим реализацию требуемых от суффиксного дерева операций на суффиксном автомате.

3.1.1. Корневая вершина

Заметим, что корневой вершине v суффиксного дерева соответствует начальное состоянию s суффиксного автомата , так как $R_T(v) = Suff(w) = R_A(s)$.

Отсюда следует, что вершина, соответствующая паре $\langle$ начальное состояние суффиксного автомата, 0 $\rangle$, является корневой вершиной суффиксного дерева.

3.1.2. Конечные вершины

Научимся быстро определять, является ли некоторая вершина суффиксного дерева конечной.

Вершина v суффиксного дерева является конечной тогда и только тогда, когда $\varepsilon \in R_T(v)$.

Пусть s — состояние суффиксного автомата, эквивалентное вершине v. Тогда вершина v является конечной тогда и только тогда, когда $\varepsilon \in R_A(s)$. Это, в свою очередь, означает, что состояние s является допускающим.

Из изложенного следует, что вершина v суффиксного дерева является конечной тогда и только тогда, когда соответствующее ей состояние s суффиксного автомата является допускающим.

Можно получить соответствующее вершине суффиксного дерева состояние суффиксного автомата и проверить, является ли оно допускающим за время $O(1)$.

3.1.3. Суффиксная ссылка

Научимся теперь быстро вычислять суффиксную ссылку из данной вершины.

Пусть вершине v сффиксного дерева соответствует пара $<s,\ len>$. Тогда из теоремы 14 следует, что:

- если строковая глубина len вершины v больше длины наименьшего представителя состояния s, то суффиксная ссылка ведет в вершину, соответствующую $\langle s, len - 1\rangle$;
- если же строковая глубина вершины равна длине наименьшего представителя состояния s, то суффиксная ссылка ведет в вершину, соответствующую $\langle suffix(s), len - 1\rangle$.

Заметим, что все указанные действия можно выполнить за $O(1)$.

3.1.4. Дуги суффиксного дерева

Научимся находить исходящую из некоторой вершины v суффиксного дерева дугу, первый символ метки которой равен c.

Пусть x — путевая метка вершины v.

Пусть вершине v суффиксного дерева соответствует состояние s суффиксного автомата.

Если в суффиксном автомате нет перехода из состояния s по символу c, то строка xc не является подстрокой строки w, что, в свою очередь, означает, что в суффиксном дереве из вершины v не выходит дуга с меткой, начинающаяся на символ c.

Теперь предположим, что в суффиксном автомате имеется переход из состояния s в состояние t по символу c. Пусть $f = fork(t)$. Из определений 31 и 33 следует, что сжатый суффиксный автомат содержит состояния s и f.

Более того, из теоремы 21 и следствия 16 следует, что в сжатом суффиксном автомате имеется переход из состояния s в состояние f по некоторому слову y, первый символ которого равен c. Из этой теоремы и следствия также следует, что $y = w[|w|-longest(s_2) \dots |w|-longest(fork(s_2))]$.

Таким образом, при наличии массивов $fork$ и $longest$, можно находить исходящую из некоторой вершины v суффиксного дерева дугу, первый символ метки которой равен c, за время, на $O(1)$ большее времени нахождения перехода по символу c, исходящего из некоторого состояния суффиксного автомата.

3.2. Задача о наибольшем общем префиксе двух суффиксов

Задача о нахождении наибольшего общего префикса двух суффиксов (LCP) формулируется следующим образом.

Задача 1 *Пусть s — некоторая строка над алфавитом Σ, заданная заранее.*

Необходимо быстро находить наибольший общий префикс строк $s_i = s[i \dots |s|]$ и $s_j = s[j \dots |s|]$.

Известны использующие суффиксные деревья алгоритмы, требующие для инициализации $O(|s|)$ времени, которые позволяют отвечать на каждый запрос за время $O(\log |s|)$ или $O(1)$.

Эта задача важна тем, что к ней можно свести следующие строковые задачи:

- задача о нахождении всех максимальных палиндромов за линейное время;
- задача о точном совпадении двух строк с джокерами;
- задача о k несовпадениях;
- задача о нахождении приблизительных палиндромов и тандемных повторов;
- задача о множественной общей подстроке за линейное время.

Решить задачу LCP с использованием суффиксного дерева можно следующим образом. На этапе инициализации построить и определенным образом обработать суффиксное дерево строки s. В дальнейшем, для получения ответа на запрос достаточно найти наименьший общий предок двух вершин.

Попытаемся решить эту задачу без использования суффиксного дерева.

Можно построить суффиксный автомат для строки s, и для ответа на некоторый запрос решать задачу о наименьшем общем предке двух вершин ациклического графа.

К сожалению, на сегодняшний день не известно ни одного алгоритма, позволяющего решить задачу о наименьшем общем предке на произвольном ациклическом графе за сублинейное время.

Заметим, что можно избавиться от суффиксного дерева, выполняя все требуемые от дерева операции над суффиксным автоматом, как описано в параграфе 3.1.

Детально рассмотрим применение этого подхода для преобразования алгоритма [6], позволяющего после линейной инициализации отвечать на каждый запрос за $O(\log |s|)$.

Фазу инициализации этого алгоритма можно реализовать следующим образом:

- построение суффиксного дерева для строки s;

- обход в глубину построенного на предыдущем этапе суффиксного дерева с запоминанием времени входа и глубины каждой вершины. При обходе дерева формируется массив, позволяющий по заданному времени получать номер соответствующей вершины;
- построение дерева отрезков над этим массивом.

Для ответа на запрос о наибольшем общем предке двух суффиксов выполняются следующие действия:

- находятся времена t_1 и t_2 входа в соответствующие этим суффиксам вершины;
- используя дерево отрезков, находится минимальная глубина вершины, которая была посещена на отрезке времени между t_1 и t_2;
- полученная глубина является длиной наибольшего общего префикса двух суффиксов.

Заметим, что фазу инициализации можно реализовать, используя суффиксный автомат. Действительно, от суффиксного дерева требуется:

- нахождение для каждого суффикса строки s соответствующую ему вершину;
- обход в глубину суффиксного дерева.

Обход в глубину суффиксного дерева можно заменить на эквивалентный ему рекурсивный обход сжатого суффиксного автомата 2.

Приведенный ниже алгоритм выполняет фазу инициализации:

```
1     public void init(String str) {
2         globalTime = 0;
3         time2depth = new int[4 * str.length() + 1];
4         suffix2time = new int[str.length() + 1];
5         SuffixAutomaton auto = new SuffixAutomaton(
            str);
6         walk(auto, auto.getStartState(), 0);
```

```
7       }
8
9       private void walk(SuffixAutomaton auto,
10                        int state,
11                        int depth)
12      {
13          if (auto.isFinal(state)) {
14              suffix2time[depth] = globalTime;
15          }
16          time2depth[globalTime++] = depth;
17          for (Edge edge: auto.getEdges(state)) {
18              int target = edge.getTarget();
19              int nextState = fork(target);
20              dfs(nextState, depth + 1 +
21                  longest(target) - longest(nextState))
                        ;
22              time2depth[globalTime++] = depth;
23          }
24      }
```

Докажем, что приведенный выше алгоритм правильно строит массивы $time2depth$ и $suffix2time$. ▷ Заметим, что в методе $walk$ выполняется рекурсивный обход сжатого суффиксного автомата. В предыдущей главе было установлено, что рекурсивному обходу сжатого суффиксного автомата соответствует рекурсивный обход суффиксного дерева. Таким образом, каждому вызову метода $walk$ соответствует некоторая вершина $node$ суффиксного дерева.

Вершина $node$ является конечной тогда и только тогда, когда состояние $state$ является допускающим. То, что вершина $node$ является конечной, означает, что ее путевая метка является суффиксом длины $depth$. В строке 14 в массив $suffix2time$ заносится время посещения вершины, соответствующей суффиксу длины $depth$.

В строке 16 в массив $time2depth$ заносится строковая глубина вершины $node$.

В цикле на строках 17 - 23 перебираются исходящие из вершины $node$ дуги. После обхода очередного ребенка посещается вершина $node$, что отмечается в массиве $time2depth$ в строке 22. $\lhd$

Из табл. 2 видно, что использование суффиксного автомата позволяет сократить время, затрачиваемое на фазу инициализации в 1.3 - 1.9 раза.

Таблица 2. Среднее время работы фазы инициализации алгоритмов на суффиксном автомате и на суффиксном дереве для решения задачи LCP. Каждый запрос выполняется за время $O(\log |w|)$. Рассматриваются строки указанной длины над алфавитом {'a', 'b'}

Длина строки	**Алг. на автомате**	**Алг. на дереве**
100000	0.163	0.234
200000	0.323	0.495
300000	0.511	0.769
400000	0.683	1.017
500000	0.852	1.289
600000	1.039	1.552
700000	1.213	1.826
800000	1.381	2.103
900000	1.563	2.381
1000000	1.735	2.631

Основная причина такой разницы времен работы сравниваемых алгоритмов состоит в том, что суффиксный автомат является более простой структурой и строится быстрее суффиксного дерева.

В связи с тем, что в результате фазы инициализации обоих алгоритмов строятся абсолютно одинаковые деревья отрезков, время выполнения запросов не изменяется. Поэтому дальнейшее сравнение алгоритмов не требуется.

Заключение

В настоящей работе подробно рассмотрена связь суффиксного дерева с суффиксным автоматом.

При этом автором разработан алгоритм построения суффиксного дерева за линейное время.

Этот алгоритм немного уступает другим алгоритмам построения суффиксного дерева (Укконена и Мак-Крейта) как по количеству используемой дополнительной памяти, так и по времени работы. Однако, он реализуется намного проще и быстрее упомянутых выше алгоритмов. Это позволяет применять его в случаях, когда требуется быстро написать правильно работающий код, затратив на это минимум усилий.

В работе также предложен метод, позволяющий избавится от использования в алгоритмах суффиксных деревьев. Суть этого метода — в изменении алгоритма таким образом, чтобы вместо операций над суффиксным деревом в нем использовались операции над суффиксным автоматом. В связи с тем, что суффиксный автомат является намного более простой в использовании структурой данных по сравнению с суффиксным деревом, применение данного метода позволяет в большинстве случаях ускорить работу алгоритма.

Для демонстрации эффективности предложенного метода в работе рассмотрена задача о нахождении наибольшего общего префикса двух суффиксов. Основная причина выбора этой задачи является то, что к ней можно свести большое число других задач на строках.

Установлено, что применение метода позволяет значительно сократить время фазы инициализации алгоритма для решения рассмотренной задачи.

В дальнейшем планируется выяснить, что выгоднее: сразу строить сжатый суффиксный автомат, или строить, а затем сжимать суффиксный автомат. Также планируется произвести сравнение количества памяти, требуемой для хранения суффиксного дерева, и памяти, требуемой для хранения суффиксного автомата с поддержкой всех операций суффиксного дерева.

Список литературы

1. *Гасфилд Д.* Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. СПб.: Невский диалект; БХВ-Петербург, 2003.

2. *Weiner P.* Linear pattern matching algorithms / Proc. of the 14th IEEE Symp. on Swithing and Automata Theory. 1973, pp.1-11.

3. *McCreight E. M.* A space-economical suffix tree construction alorithm // J. ACM. 1976. Vol 23, pp. 262-272.

4. *Ukkonen E.* Online construction of suffix-trees // Algoritmica. 1995. Vol. 14, pp. 249-260.

5. *Lothaire M.* Applied Combinatorics on Words // Encyclopedia of Mathematics and its Applications, 2005. Vol. 90. Cambridge University Press, Cambridge.

6. *Bender M., Farach-Colton M.* The LCA Problem Revisited / LATIN 2000, pp. 88-94.

7. *Хопкрофт Дж., Мортвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.

8. *Кормен Т., Лейзерсон Ч., Ривест Л.* Алгоритмы: построение и анализ. М.: МЦНМО, 2000.

9. *Sartaj Sahni Dr.* Data Structures, Algorithms, & Applications in Java. Suffix Trees. CISE Department Chair at University of Florida. http://www.cise.ufl.edu/s̃ahni/dsaaj/enrich/c16/suffix.htm

10. *Edelkamp S.* Suffix tree // Dictionary of Algorithms and Data Structures. U.S. National Institute of Standards and Technology. 2007. http://www.nist.gov/dads/HTML/suffixtree.html

11. *Blumer A., Blumer J., Ehrenfeucht A., Hausler D., McConnel R.* Linear size finite automata for the set of all subwords of a word: an outline of results // Bull. Eur. Assoc. Theoret. Commput. Sci. 1983. Vol 21, pp. 12-20.

12. *Blumer A., Blumer J., Ehrenfeucht A., Hausler D., McConnel R.* The smallest automaton recognizing the subwords of a text // Bull. Eur. Assoc. Theoret. Commput. Sci. 1985. Vol 40(1), pp. 31-55.

13. *Eilenberg S.* Automata, Languages, and Machines. Vol A. Academic Press. 1974.

14. *Kuich W., Salomaa A.* Semirings, Automata, Languages. Springer-Verlag. 1986.

15. *Alonso L., Rémy J. L., Schott R.* A linear-time algorithm for the generation of trees // Algorithmica. 1997. Vol 17(2), pp. 162-182.

16. *Devroye L., Szpankowski W., Rais B.* A note of the height of suffix trees // SIAM J. Comput. 1992. Vol. 21, pp. 48-53.

17. *Farach M.* Optimal suffix tree construction with large alphabets // In 38th Foundations of Computer Science (FOCS). 1997, pp. 137-143.