Бор
Суффиксный бор
Суффиксный массив

Задача поиска заданной строки в множестве строке

Предположим, что есть множество строк

$$S = \{S_0, S_1, \ldots, S_{n-1}\}$$

из букв латинского алфавита Σ и необходимо быстро проверить, есть ли среди них некоторая заданная строка

$$A = (a_0, a_2, \ldots, a_{l-1}).$$

Если поместить все строки из множества $\boldsymbol{S} = \{S_0, S_1, \dots, S_{n-1}\}$ в **массив** и осуществлять поиск строки $\boldsymbol{A} = (a_0, a_2, \dots, a_{l-1})$ последовательно просматривая элементы массива, то **время поиска строки** A $-$

$$\mathbf{O}(|\boldsymbol{A}| \cdot \boldsymbol{n}),$$

где n – количество строк в S.

При этом требуемая **памят**ь $-$ $\boldsymbol{O}(|\boldsymbol{S_0}| + |\boldsymbol{S_1}| + \cdots + |\boldsymbol{S_{n-1}}|)$.

Если для хранения строк из множества $S = \{S_0, S_1, \dots, S_{n-1}\}$ использовать сбалансированное бинарное поисковое дерево, то время **поиска строки** A $-$

$$\mathbf{O}(|\boldsymbol{A}| \cdot \log \boldsymbol{n}),$$

где n – количество строк в S.

Существуют структуры данных, которые позволяют выполнять поиск строки более эффективно, например, за время

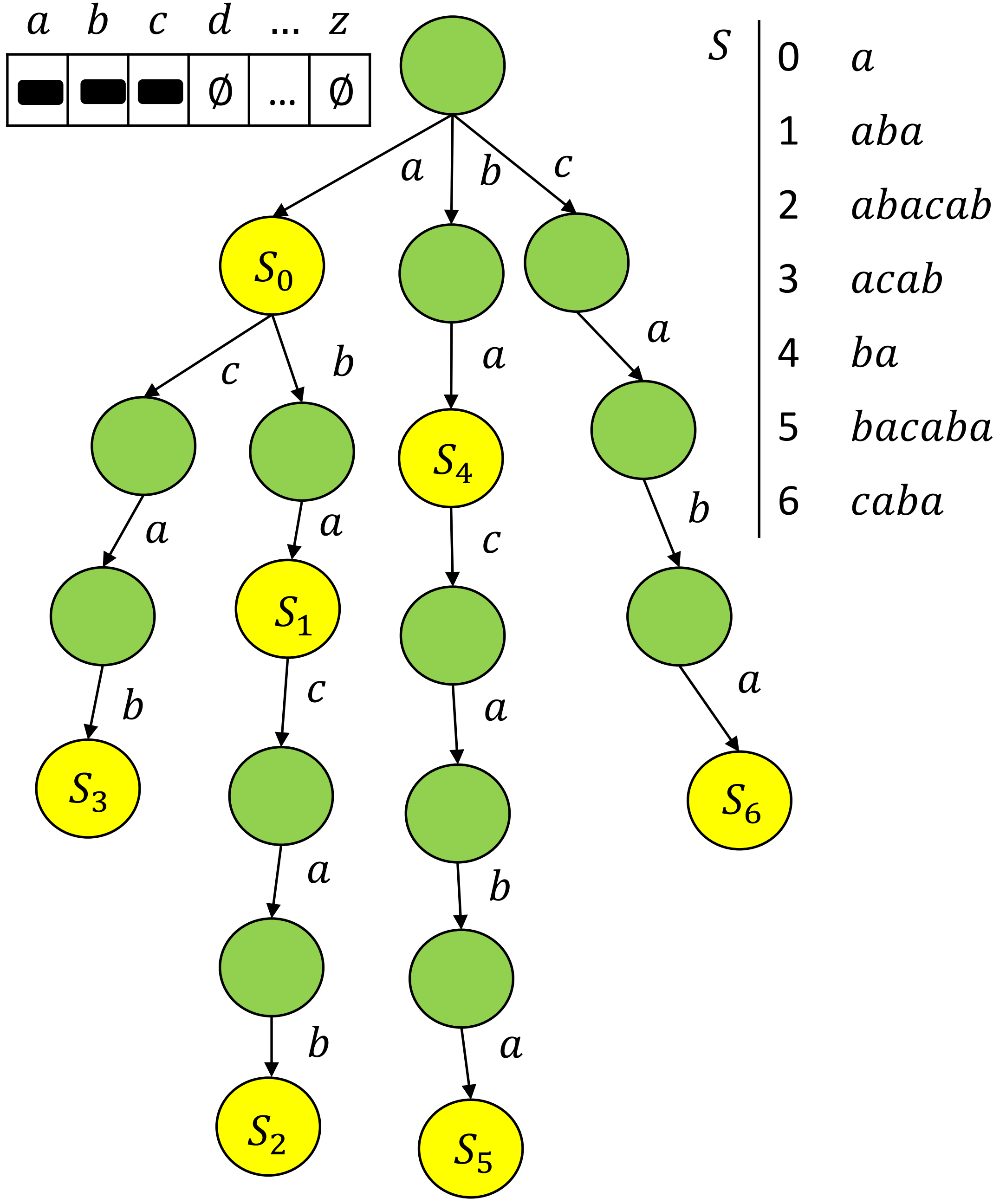
$$\mathbf{O}(|\boldsymbol{A}|)$$

(время поиска не зависит от количества строк в S).

Одной из таких структур данных является **бор**.

Бор

(*англ.* trie, луч, нагруженное дерево)



$A = aba$

S

0	a
1	aba
2	$abacab$
3	$acab$
4	ba
5	$bacaba$
6	$caba$

Разобьем слова из множества $S = \{S_0, S_1, \dots, S_{n-1}\}$ на группы в соответствии с первыми символами слов.

Пусть S' - группа, у слов которой первый символ совпадает с первым символом строки A. Сокращаем область поиска до множества S', исключив из рассмотрения те строки, которые туда не попадают.

Заведем массив, в качестве индексов которого будут выступать символы алфавита, а в самом массиве будут храниться сами строки.

S'

$A = (\underline{\mathbf{a}}, b, a)$

a	b	c	d
a aba $abacaba$ $acab$	ba $bacaba$	$caba$	

Удаляем из всех слов множества S' и слова A первый символ и повторяем процедуру.

S'

1	ba
2	$bacaba$
3	cab

$A' = (\cancel{a,} \underline{\mathbf{b}}, a)$

a	b	c	d
	ba $bacaba$	cab	

S'

1	a
2	$acaba$

$A' = (\cancel{a, b,} \underline{\mathbf{a}})$

a	b	c	d
a $acaba$			

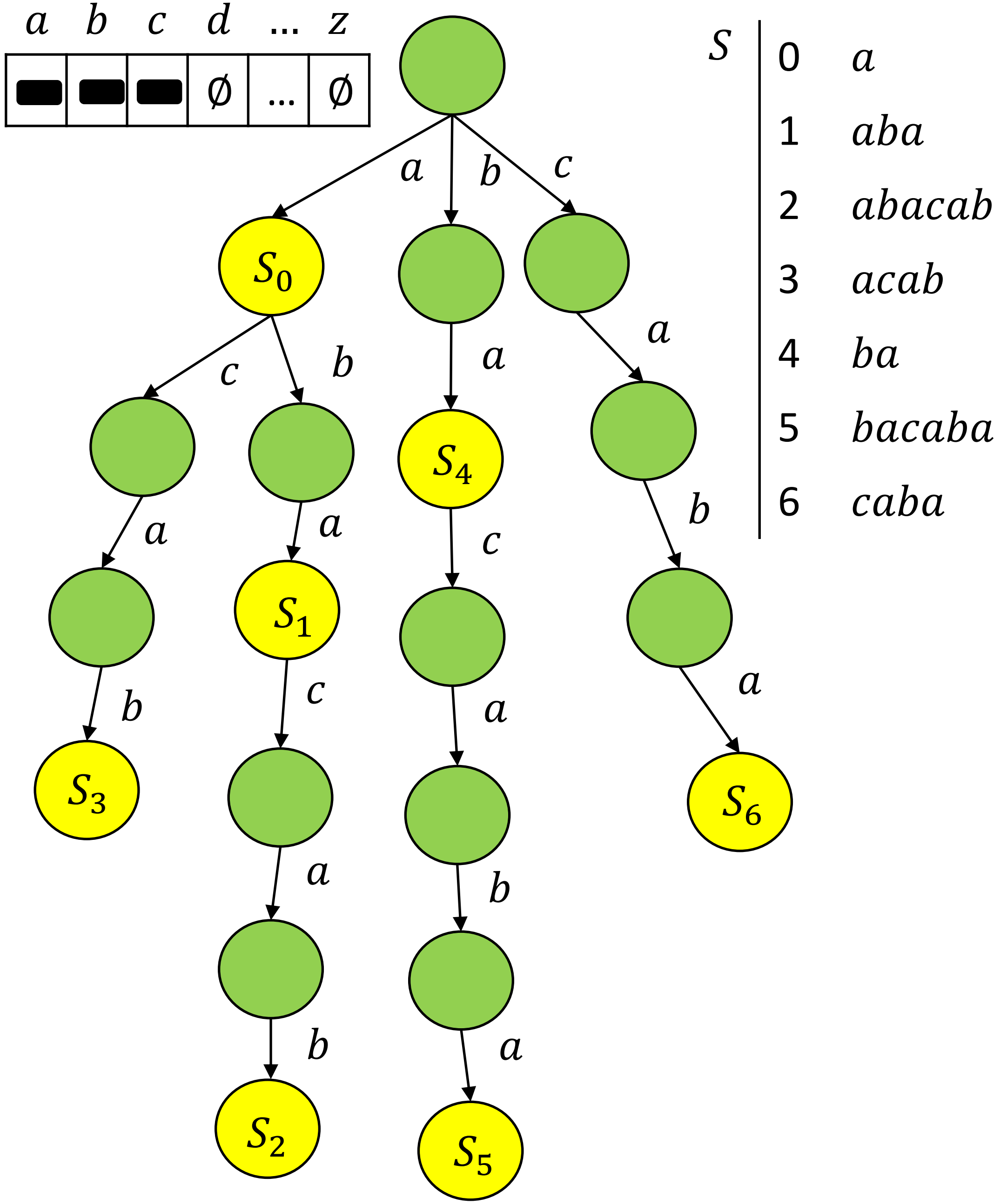
Если искомая строка A'состоит из одного символа, то достаточно проверить наличие в множестве S' строки длины 1.

Приведенное иерархическое разбиение строк на множества можно изобразить в виде древовидной структуры.

Бор

| специализированная древовидная структура данных, предназначенная для хранения слов некоторого алфавита Σ.

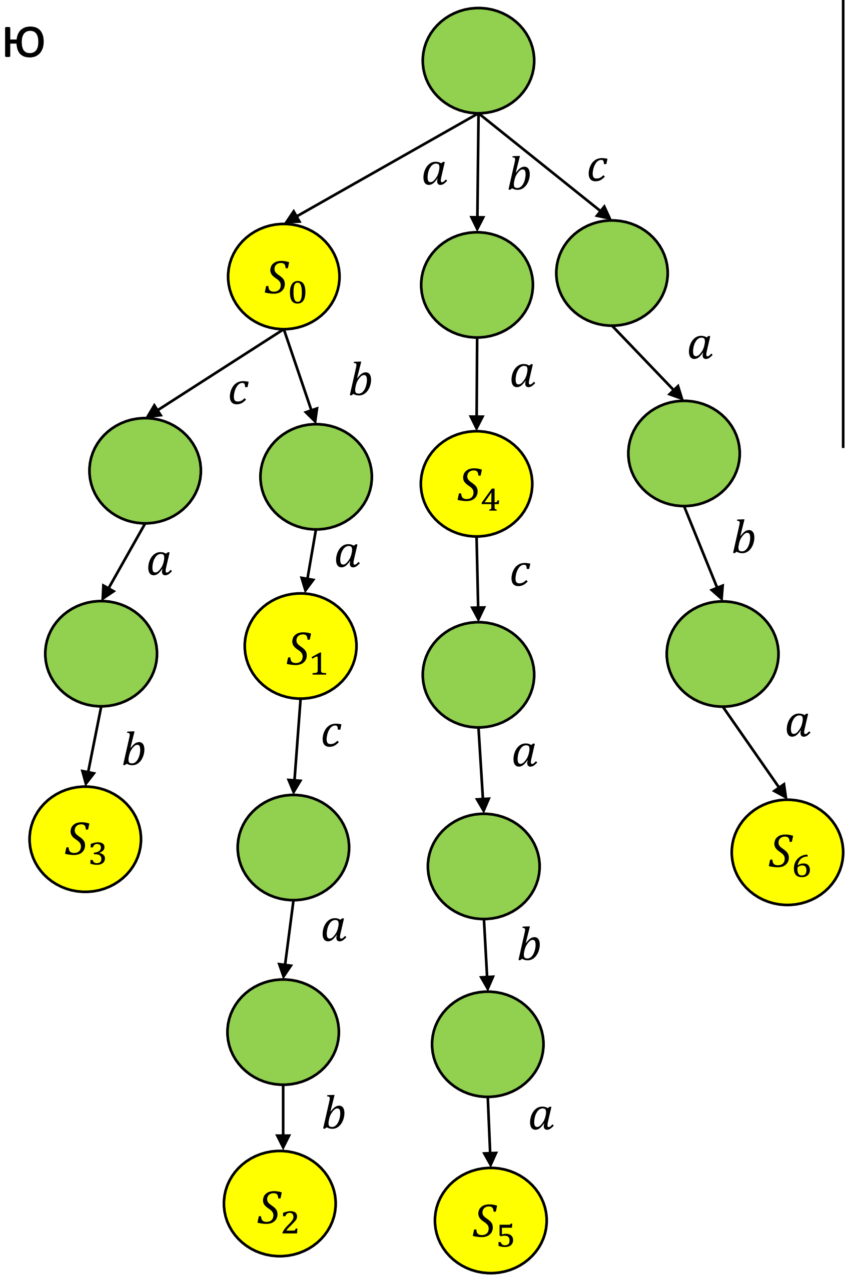
- **вершина дерева** содержит
  - информацию о вершинах, в которые можно перейти по каждому допустимому символу (если переход по дуге с некоторым символом не возможен, то результат перехода будем обозначать $\emptyset$);
  - пометку: терминальная или нет (вершина v – терминальная, если строка, которая определяется последовательностью символов, встречающихся на дугах в пути от корня к v, есть в множестве S);
  - терминальная вершина может хранить индекс слова в S;
- **дуге** ставится в соответствие символ строки;



Если **выписать все символы на пути из корня в терминальную вершину**, то получим некоторую строку (слово) из множества $\boldsymbol{S}$.

Листья дерева соответствуют строкам из $\boldsymbol{S}$.

Если некоторая строка из $\boldsymbol{S}$ является префиксом другой строки этого множества, то ей будет соответствовать внутренняя вершина дерева ($S_1 = \boldsymbol{aba}$, $S_2 = \boldsymbol{aba}cab$).



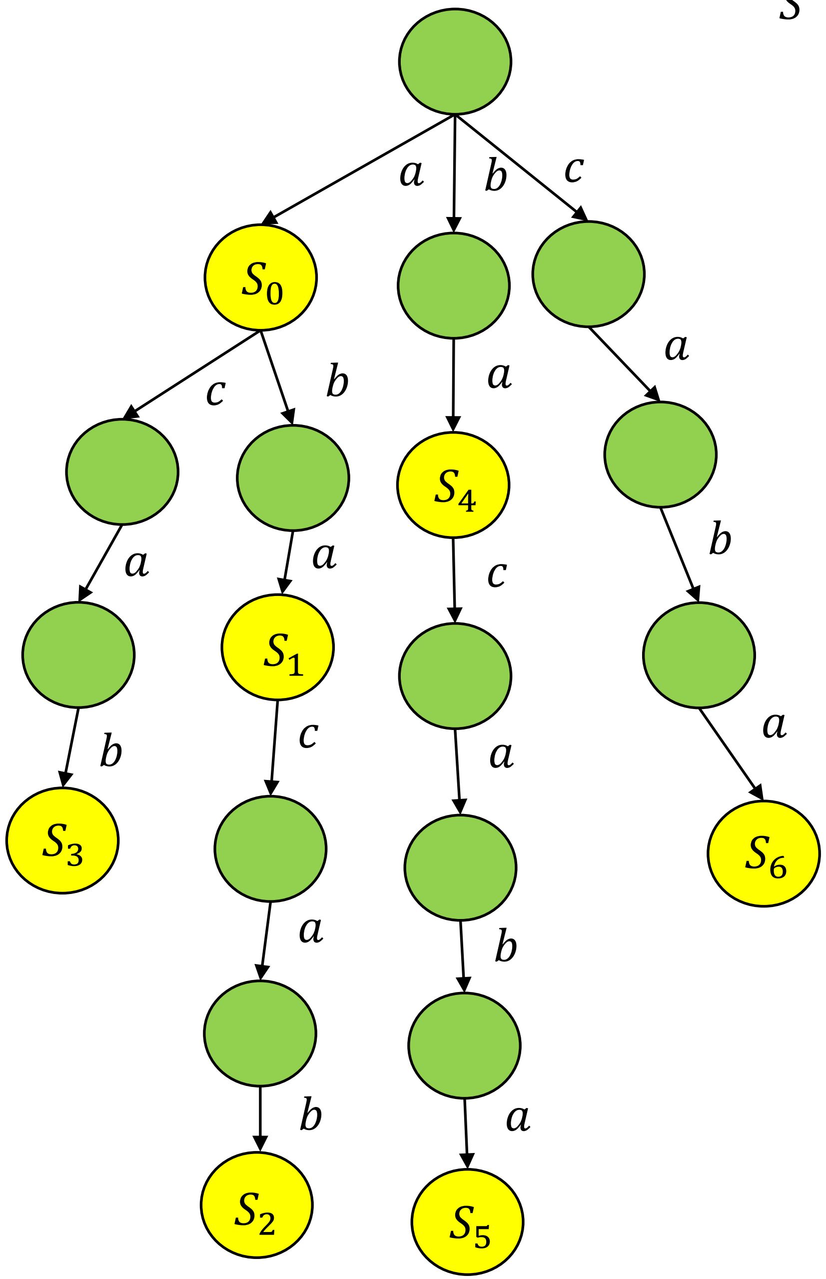
S		
	0	a
	1	aba
	2	$abacab$
	3	$acab$
	4	ba
	5	$bacaba$
	6	$caba$

Для **поиска строки $\boldsymbol{A}$ в боре** будем последовательно спускаться из корня дерева по дугам, соответствующим символам строки $\boldsymbol{A}$, пока строка $\boldsymbol{A}$ не закончится и в боре существует дуга, соответствующая текущему состоянию строки $\boldsymbol{A}$.

Если в результате спуска:

(1) попадём в терминальную вершину и дойдём до конца строки $\boldsymbol{A}$, то **искомое слово найдено**;

(2) если остановимся в нетерминальной вершине либо во время спуска не найдём дуги в дереве, соответствующей текущему символу строки $\boldsymbol{A}$, то делаем вывод о том, что **строка $\boldsymbol{A}$ в** множестве $\boldsymbol{S}$ **отсутствует**.



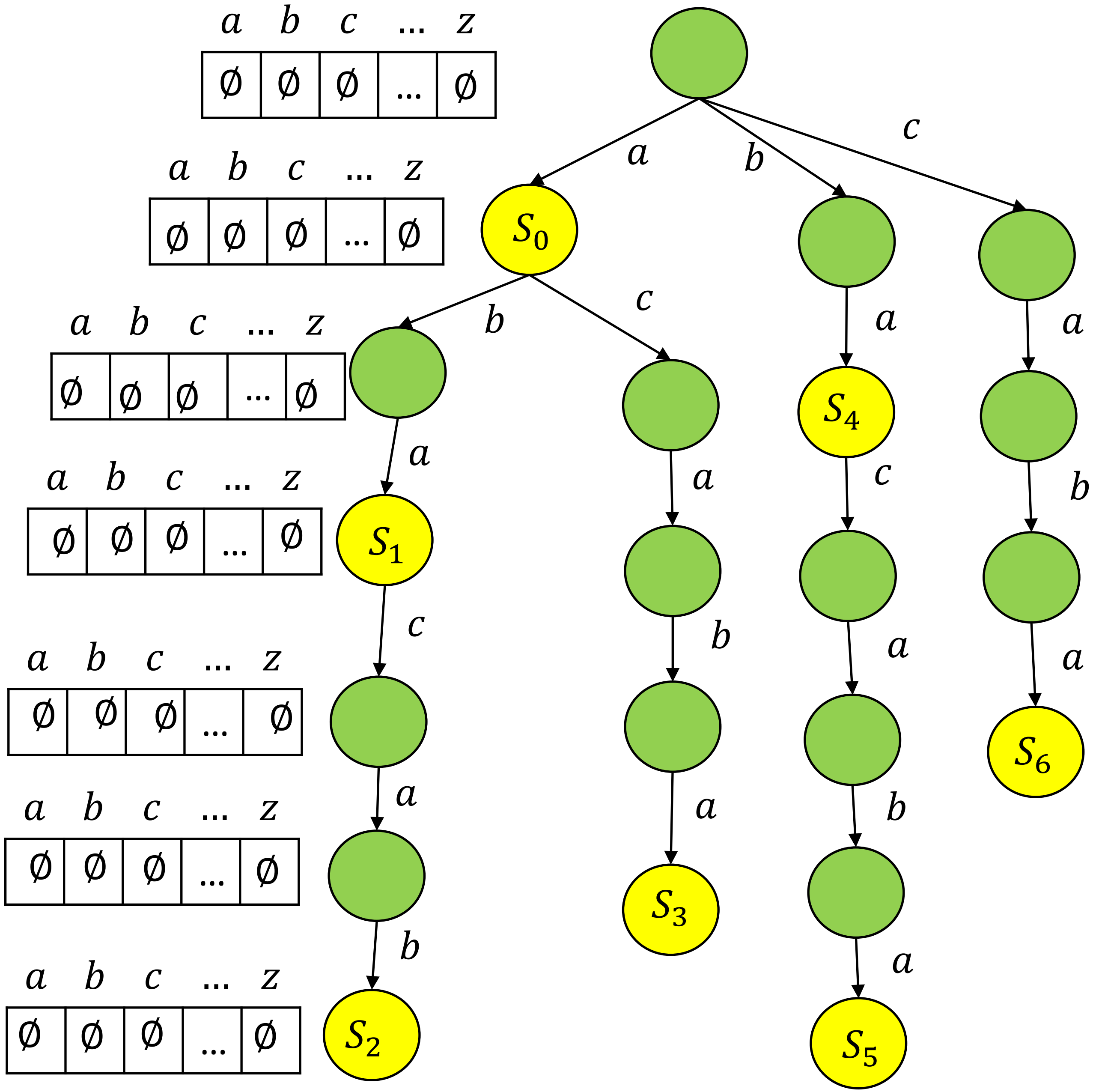
S	
0	a
1	aba
2	$abacab$
3	$acab$
4	ba
5	$bacaba$
6	$caba$

Добавление строки

Как и во время поиска, спускаемся по дереву, начиная с корня. Если во время спуска мы попадаем в ситуацию, что надо перейти по несуществующей дуге, то добавляем к дереву новую вершину и ведущую к ней дугу с соответствующей пометкой.

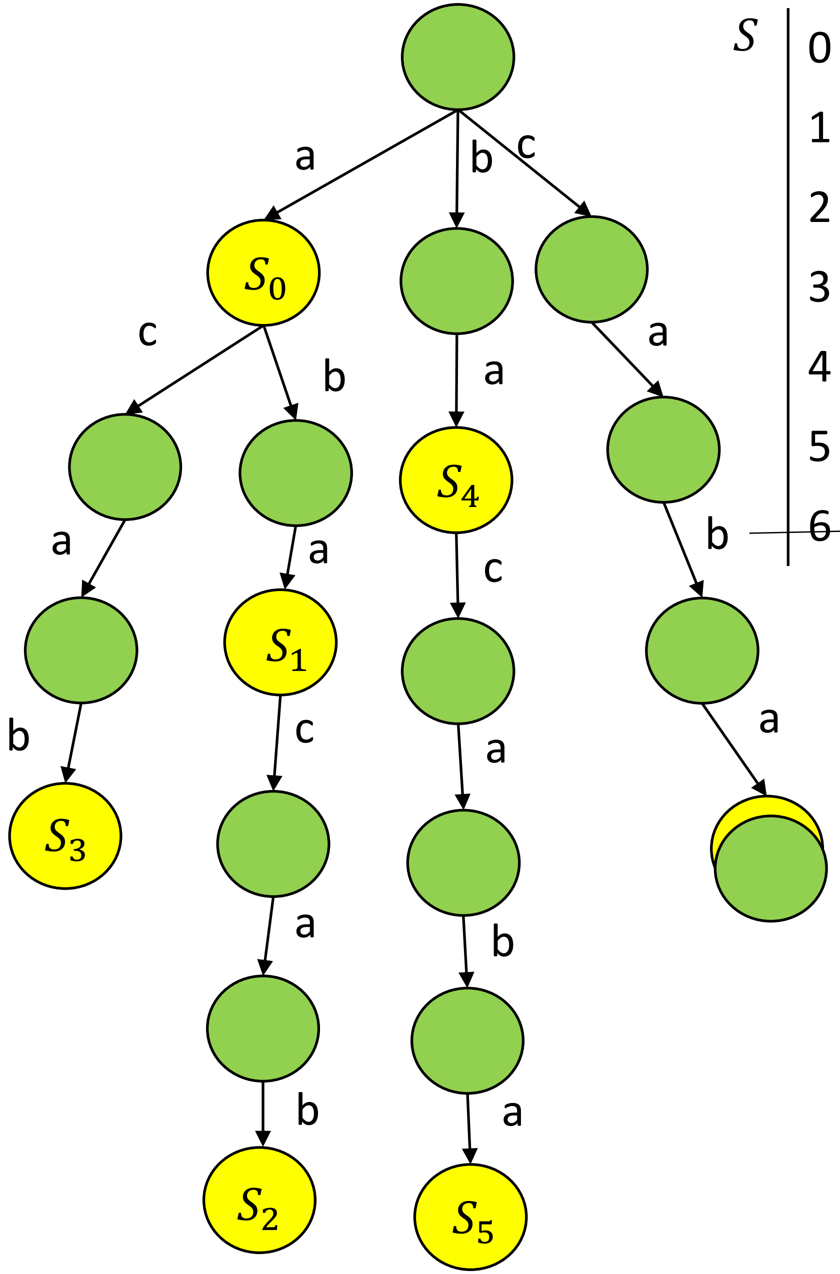
Движение осуществляем, пока не пройдем все символы добавляемой строки.

S	
0	a
1	aba
2	$abacaba$
3	$acaba$
4	ba
5	$bacaba$
6	$caba$



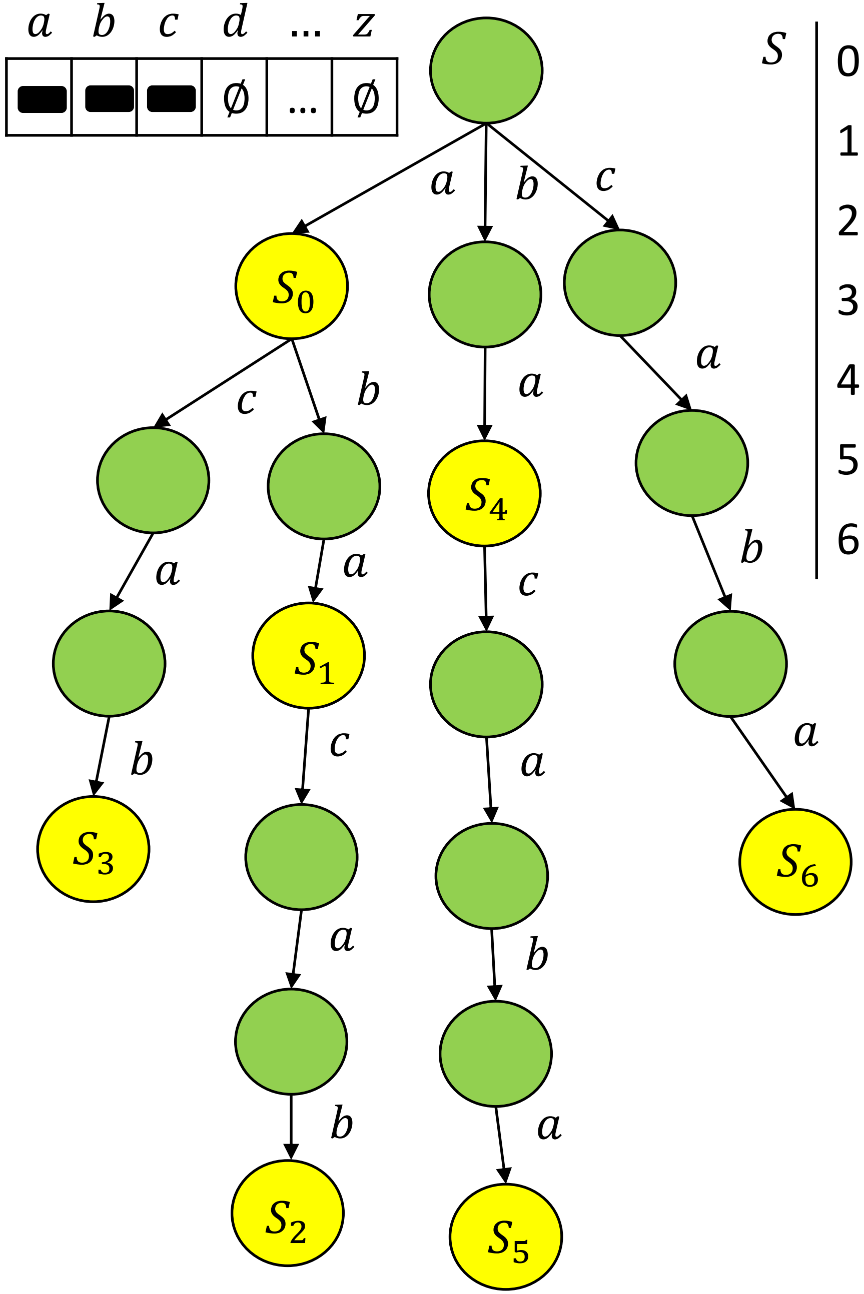
Удалить строку

из бора можно «*по ленивому*» – снять метку с соответствующей терминальной вершины.



S	
0	a
1	aba
2	$abacab$
3	$acab$
4	ba
5	$bacaba$
~~6~~	~~$caba$~~

ОЦЕНКИ



S	
0	a
1	aba
2	$abacab$
3	$acab$
4	ba
5	$bacaba$
6	$caba$

Время **построения** структуры данных бор для множества строк из $S = \{S_0, S_1, \dots, S_{n-1}\}$ есть

$$O\left(\sum_{i=0}^{n-1} |S_i|\right),$$

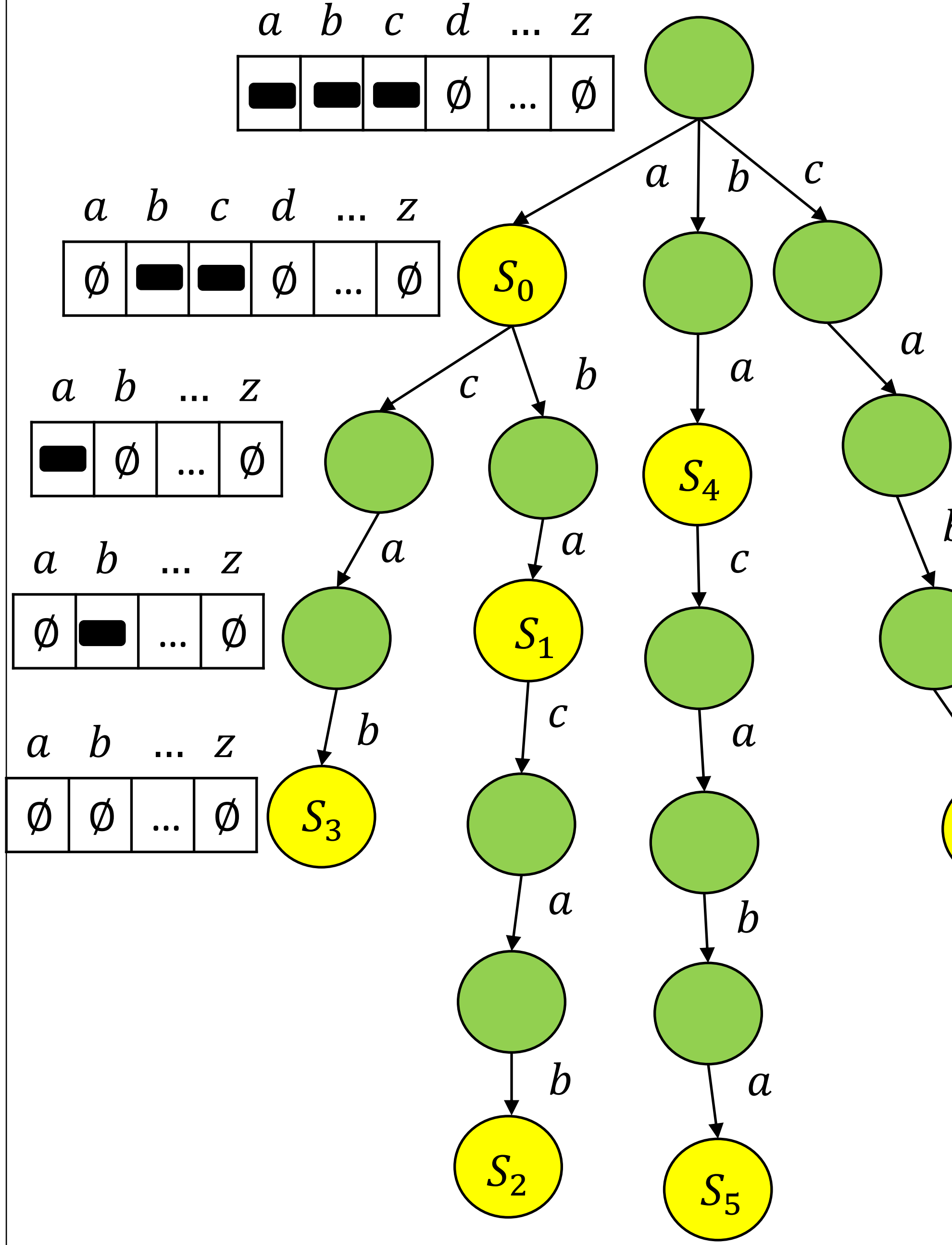
требуемая **память**:

$$O\left(|\Sigma| \cdot \sum_{i=0}^{n-1} |S_i|\right),$$

где n – число строк в множестве S,
$|S_i|$ – длина i-ой строки,
$|\Sigma|$ - размер алфавита.

Время **добавления/поиска** строки $\boldsymbol{A = \{a_0, a_2, \dots, a_{l-1}\}}$ зависит от длины строки A и не зависит от количества строк в боре:

$$O(l).$$



S	
0	a
1	aba
2	$abacab$
3	$acab$
4	ba
5	$bacaba$
6	$caba$

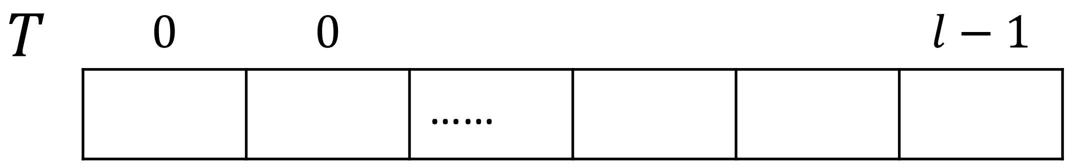
Множество строк $S = \{S_0, S_1, \dots, S_{n-1}\}$	**Строка** $A = (a_0, a_1, \dots, a_{l-1})$	**Определить, есть ли строка A в S?**

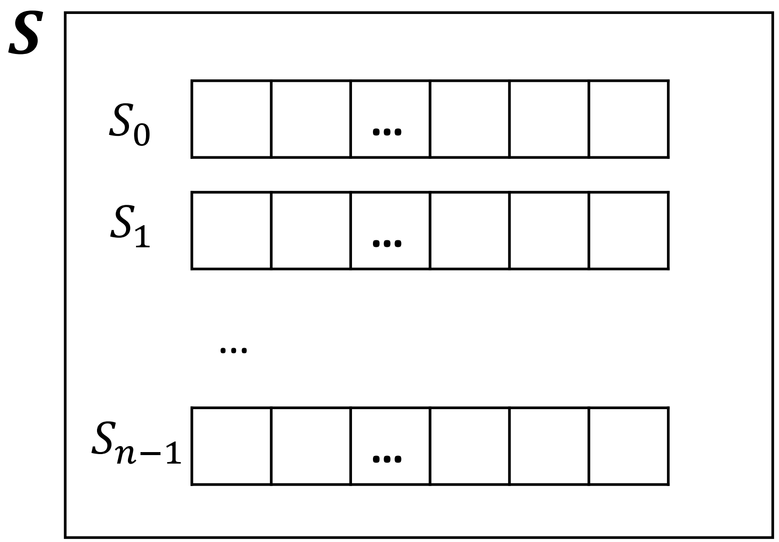
Структура данных	Время построения	Память	Время поиска
Массив	$O\left(\sum_{i=0}^{n-1} \lvert S_i \rvert\right)$	$O\left(\sum_{i=0}^{n-1} \lvert S_i \rvert\right)$	$O(\lvert A \rvert \cdot n)$
Сбалансированное бинарное поисковое дерево	$O(n \cdot \log n \cdot l_{max})$, $l_{max} = \max_{i=0,..,n-1} \lvert S_i \rvert$	$O\left(\sum_{i=0}^{n-1} \lvert S_i \rvert\right)$	$O(\lvert A \rvert \cdot \log n)$
Бор	$O\left(\sum_{i=0}^{n-1} \lvert S_i \rvert\right)$	$O\left(\lvert \Sigma \rvert \cdot \sum_{i=0}^{n-1} \lvert S_i \rvert\right)$	$O(\lvert A \rvert)$

Задача поиска множества образцов в строке в режиме on-line

- Пусть у нас есть текст $\boldsymbol{T} = (t_0, t_1, \dots, t_{l-1})$, и несколько образцов, которые поступают в режиме *on-line* из множества $\boldsymbol{S} = \{S_0, S_1, \dots, S_{n-1}\}$.

Найти все подстроки текста $\boldsymbol{T}$, которые совпадали бы с одним из образцов (т.е. необходимо определить, встречается ли образец из S в качестве подстроки в тексте T).





Задача поиска множества образцов в строке в режиме реального времени

Решить задачу можно эффективно, используя такие структуры данных, как

- **суффиксный бор**
- **суффиксный массив**

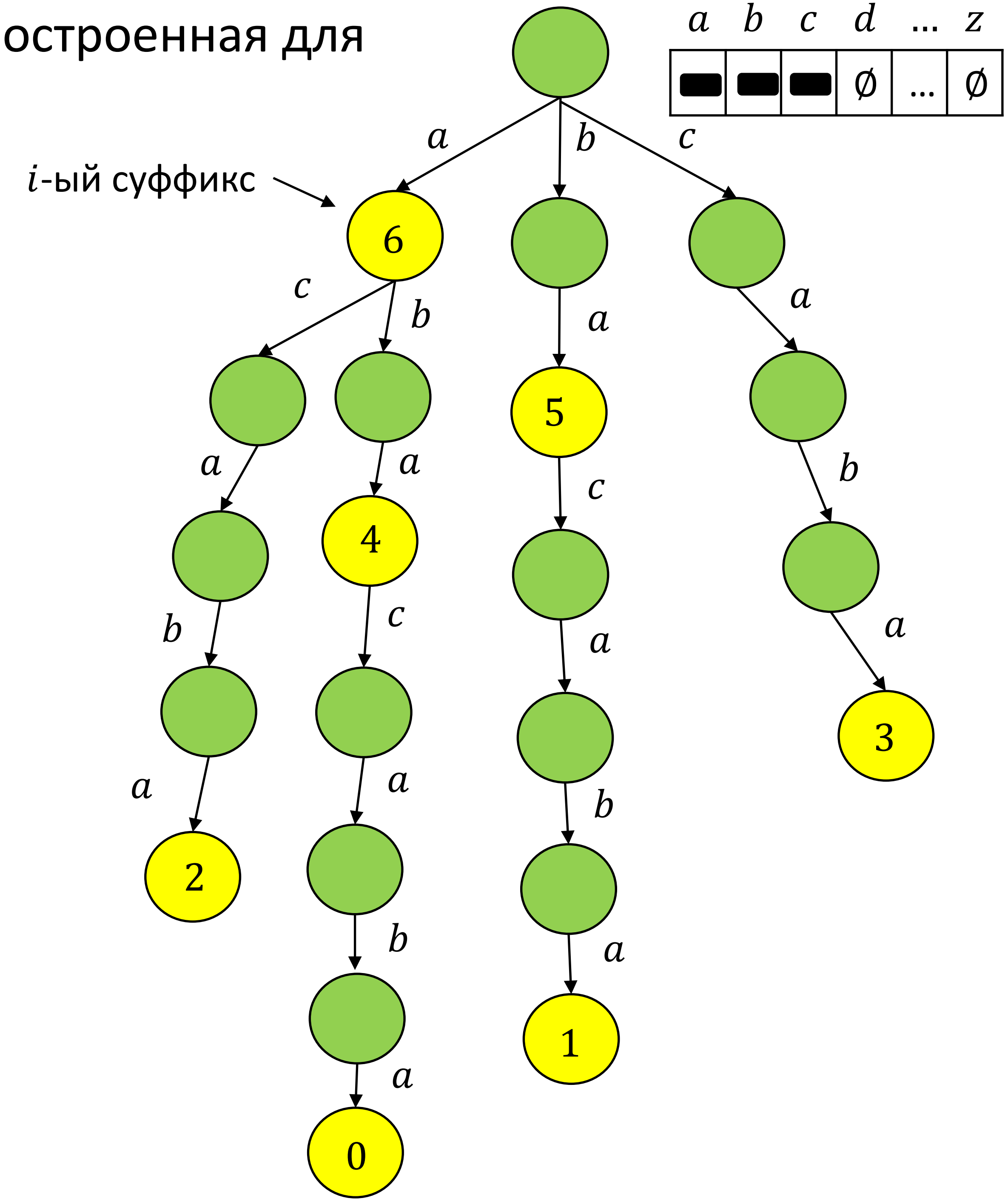
Суффиксный бор

Суффиксный бор - структура данных бор, построенная для всех суффиксов текста $\mathrm{T} = (t_0, t_1, \dots, t_{l-1})$.

$\boldsymbol{T = abacaba}$

0	1	2	3	4	5	6
a	b	a	c	a	b	a

0-й	$abacaba$
1-й	$bacaba$
2-й	$acaba$
3-й	$caba$
4-й	aba
5-й	ba
6-й	a



Время построения : $\mathrm{O}(l^2)$,

требуемая память: $\mathrm{O}(|\Sigma| \cdot l^2)$,

где $l = |T|$,$|\Sigma|$ - размер алфавита.

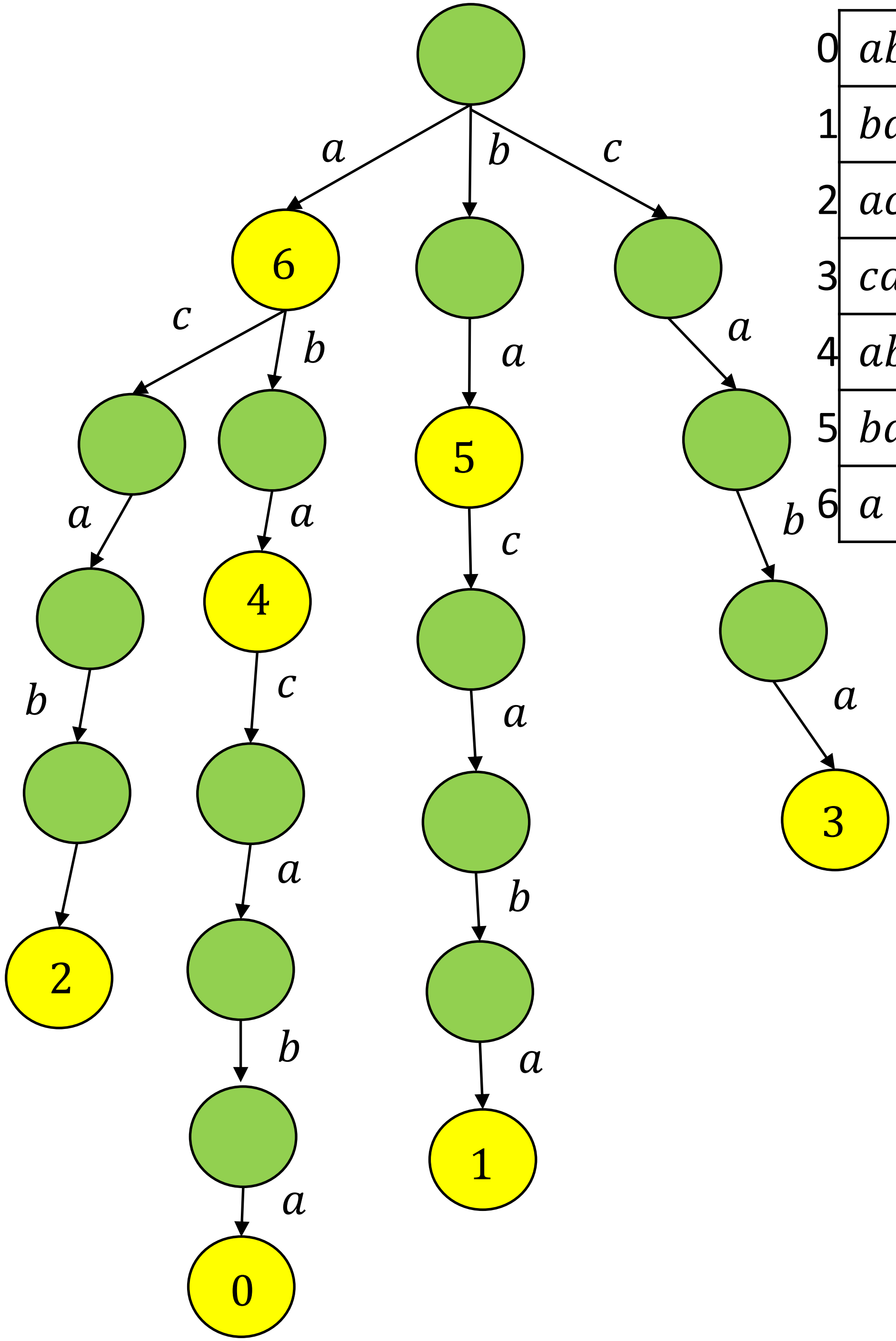
Если в **суффиксном боре** находится некоторый суффикс, то все префиксы этого суффикса также находятся в боре.

Если некоторая строка $\boldsymbol{S_i}$ встречается в качестве подстроки в строке $\boldsymbol{T}$, то она является префиксом какого-то суффикса строки $\boldsymbol{T}$. Поэтому суффиксный бор можно использовать для решения задачи поиска множества образцов в строке в режиме on-line.

Время поиска множества образцов в строке:

$$\mathbf{O}\left(\sum_{i=0}^{n-1}|\boldsymbol{S_i}|\right) = \mathbf{O}(n \cdot l_{max}),$$

где $l_{max} = \max\limits_{i=0,..,n-1} |S_i|$.



0	*abacaba*
1	*bacaba*
2	*acaba*
3	*caba*
4	*aba*
5	*ba*
6	*a*

Суффиксный массив

Задан фиксированный текст $\mathrm{T} = (t_0, t_1, \dots, t_{l-1})$	**On-line поступают образцы из множества** $S = \{S_0, S_1, \dots, S_{n-1}\}$	**Определить, встречается ли образец из S в качестве подстроки в тексте** T

Структура данных	Время построения	Память	Время поиска
Суффиксный массив	$O(\lvert T\rvert \cdot log^2\lvert T\rvert)$	$O(\lvert T\rvert)$	$O\left(log\lvert T\rvert \cdot \sum_{i=1}^{n-1}\lvert S_i\rvert\right)$
Суффиксный бор	$O(\lvert T\rvert^2)$	$O(\lvert T\rvert^2 \cdot \lvert\Sigma\rvert)$	$O\left(\sum_{i=0}^{n-1}\lvert S_i\rvert\right)$

Структура данных **суффиксный массив** была разработана в **1989** году .

Юджин Уимберли «Джин» Майерс-младший
Eugene Wimberly «Gene» Myers, Jr.

Дата рождения	31 декабря 1953
Место рождения	•Бойсе, США
Страна	• США
Научная сфера	информатика

Уди Манбер
Udi Manber

Место рождения:	Кирьят-Хаим, Израиль
Страна	• США
Научная сфера	информатика
Должность	вице-президент Google по разработкам

Суффиксный массив строки $T = (t_0, t_1, \dots, t_{l-1})$ это последовательность **лексикографически отсортированных суффиксов** строки T (очевидно, что достаточно хранить только индексы суффиксов).

i	0	1	2	3	4	5	6
T	a	b	a	c	a	b	a

$l = 7$

перестановка индексов суффиксов, которая задаёт порядок суффиксов в порядке лексикографической сортировки

все суффиксы строки T

0-й	$abacaba$
1-й	$bacaba$
2-й	$acaba$
3-й	$caba$
4-й	aba
5-й	ba
6-й	a

лексикографическая сортировка суффиксов →

a
aba
$abacaba$
$acaba$
ba
$bacaba$
$caba$

→

6-й	a
4-й	aba
0-й	$abacaba$
2-й	$acaba$
5-й	ba
1-й	$bacaba$
3-й	$caba$

суффиксный массив $P[0..l-1]$ →

0	6
1	4
2	0
3	2
4	5
5	1
6	3

Пусть есть фиксированный текст $\mathrm{T} = (t_0, t_1, \dots, t_{l-1})$:

	0	1	2	3	4	5	6
T	a	b	a	c	a	b	a

На вход в режиме реального времени поступают образцы из множества $\mathrm{S} = \{S_0, S_1, \dots, S_{n-1}\}$, и нам необходимо эффективно определять, встречается ли образец $S' \in S$ в качестве подстроки в тексте T (*on-line* версия задачи).

S'

a	b

Как, используя суффиксный массив, эффективно решить эту задачу?

	0	1	2	3	4	5	6
T	a	b	a	c	a	b	a

	0	1
$S' \in S$	a	b

1) Построили по тексту T суффиксный массив $\boldsymbol{P}$

0	6	a
1	4	aba
2	0	$abacaba$
3	2	$acaba$
4	5	ba
5	1	$bacaba$
6	3	$caba$

2) Если строка $\boldsymbol{S'}$ является подстрокой $\boldsymbol{T}$, то она является префиксом какого-то суффикса, поэтому дихотомией (*LowerBound,UpperBound*) по суффиксному массиву $\boldsymbol{P}$ (осуществляем поиск диапазона суффиксов, у которых префикс совпадает с образцом $\boldsymbol{S'}$).

S': $\boldsymbol{a}$ | (пусто)

0	6	$\boldsymbol{a}$
1	4	$\boldsymbol{a}ba$
2	0	$\boldsymbol{a}bacaba$
3	2	$\boldsymbol{a}caba$

$L = 0, \quad R = n$

$L = LowerBound(L, R, s'[0])$

$R = UpperBound(L, R, s'[0])$

$L = 0, R = 4$

если в диапазоне у суффиксов, у которых первый символ совпадает, удалить этот символ, то по следующему символу эти префиксы в этом диапазоне также упорядочены лексикографически

S': ~~a~~ | **b**

0	6	$\cancel{a}$
1	4	$\cancel{a}ba$
2	0	$\cancel{a}bacaba$
3	2	$\cancel{a}caba$

$L = LowerBound(L, R, s'[1])$

$R = UpperBound(L, R, s'[1])$

1	4	$\cancel{a}ba$
2	0	$\cancel{a}bacaba$

$L = 1, R = 3$

Время поиска вхождений образца $S' \in S$ в $T = (t_0, t_1, \dots, t_{l-1})$

$$\Theta(|S'| \cdot log|T|) = \Theta(|S'| \cdot \log l).$$

Как построить суффиксный массив?

Алгоритм построения суффиксного массива для $\boldsymbol{T = (t_0, t_1, \dots, t_{l-1})}$ за время

$$\boldsymbol{\Theta}\left(|T|^{\boldsymbol{2}} + |T| \cdot |\Sigma|\right)$$

заключается, например, в непосредственном упорядочивании всех суффиксов строки $\boldsymbol{T}$ алгоритмом **лексикографической сортировки**.

Время работы
алгоритма лексикографической
сортировки

$$\Theta\left(\sum_{i=0}^{l-1} \left|t^j\right| + l_{max} \cdot |\Sigma|\right), \text{где}$$

l – число кортежей,
l_{max} – длина самого длинного кортежа,
$|\Sigma|$ – число различных символов в кортежах.

Как построить суффиксный массив эффективнее?

Если для сортировки суффиксов строки $\boldsymbol{T} = (t_0, t_1, \dots, t_{l-1})$ использовать, например, **сортировку слиянием** (***merge sort***), то время работы алгоритма построения суффиксного массива будет зависеть от того, как будут сравниваться суффиксы.

Непосредственное сравнение двух суффиксов приведет к тому, что время работы алгоритма построения суффиксного массива:

$$\boldsymbol{O}\big(|\boldsymbol{T}| \cdot (|\boldsymbol{T}| \cdot \boldsymbol{log}|\boldsymbol{T}|)\big) = \boldsymbol{O}\big(|\boldsymbol{T}|^2 \cdot \boldsymbol{log}|\boldsymbol{T}|\big).$$

Как научиться сравнивать строки быстрее, например, за константное время?

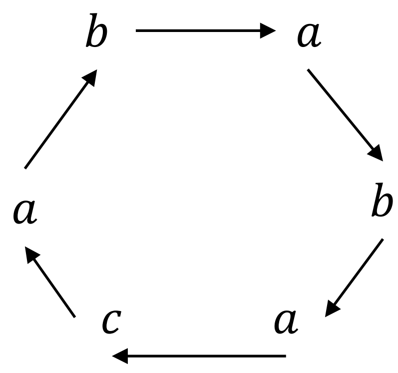
Алгоритм
построения суффиксного массива
строки $T = (t_0, t_1, \dots, t_{l-1})$ за время

$$\mathrm{O}(|T| \cdot log^2|T|) = \mathbf{O}(\boldsymbol{l} \cdot \mathbf{log}^2\, \boldsymbol{l})$$

Циклический сдвиг строки T

> это строка, которая получается из исходной строки T путем перемещением её первых символов в конец строки.

$T = abacaba\$$



циклический сдвиг длины 1	$bacabaa$
циклический сдвиг длины 2	$acabaab$
циклический сдвиг длины 3	$cabaaba$
циклический сдвиг длины 4	$abaabac$
циклический сдвиг длины 5	$baabaca$
циклический сдвиг длины 6	$aabacab$
циклический сдвиг длины 6	$abacaba$

лексикографически минимальный циклический сдвиг

Если к строке T добавить $\$$ (символ, который меньше всех символов строки), выполнить лексикографическую сортировку всех циклических сдвигов $T + \$$, затем удалить из каждого циклического сдвига строки суффикс, который начинается с $\$$, то получим лексикографическую сортировку всех суффиксов строки T.

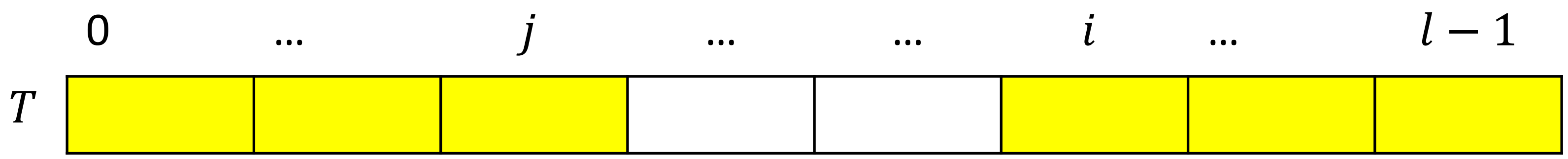
$T = \boldsymbol{abacaba}$

$T + \$ = \ \boldsymbol{abacaba} + \$$

Циклические сдвиги $T + \$$	Лексикографическая сортировка	Лексикогр. упорядоченные суффиксы строки T
$bacaba\$a$	$\$abacaba$	
$acaba\$ab$	$a\$abacab$	$\boldsymbol{a}$
$caba\$aba$	$aba\$abac$	$\boldsymbol{aba}$
$aba\$abac$	$abacaba\$$	$\boldsymbol{abacaba}$
$ba\$abaca$	$acaba\$ab$	$\boldsymbol{acaba}$
$a\$abacab$	$ba\$abaca$	$\boldsymbol{ba}$
$\$abacaba$	$bacaba\$a$	$\boldsymbol{bacaba}$
$abacaba\$$	$caba\$aba$	$\boldsymbol{caba}$

Поэтому научимся эффективно упорядочивать циклические сдвиги строки.

Циклическая подстрока $t[i..j],\ i > j$ определяется, как $t[i..l-1] + t[0..j]$.



Пусть $l = |T|,\ 2^{k-1} < l \le 2^k$.

Тогда
$$2^{k-1} < l$$
$$k < log_2 l + 1$$
$$k = \lceil log_2 l \rceil$$

Выполним $\boldsymbol{k = \lceil log_2 l \rceil}$ **фаз сортировки циклических подстрок**.

На каждой фазе выполняется лексикографическая сортировка циклических подстрок одинаковой длины.

На i-ой фазе сортируются циклические подстроки длины $2^i, 0 \le i \le k = \lceil log_2 l \rceil$.

На $\boldsymbol{j}$ – ой фазе поддерживаются следующие массивы:

$\boldsymbol{P}^{\boldsymbol{j}}$ – перестановка индексов циклических подстрок длины 2^j, которая задаёт порядок циклических подстрок длины 2^j в порядке лексикографической сортировки;

$\boldsymbol{C}^{\boldsymbol{j}}$– массив классов эквивалентности (элементы массива - целые числа ≥ 0), где

- ✓ $c^j[i]$ – номер класса эквивалентности для циклической подстроки длины 2^j, начинающейся в позиции i;
- ✓ если две циклические строки совпадают, то у них один класс эквивалентности;
- ✓ если одна циклическая подстрока лексикографически меньше другой, то номер класса она получит меньший.

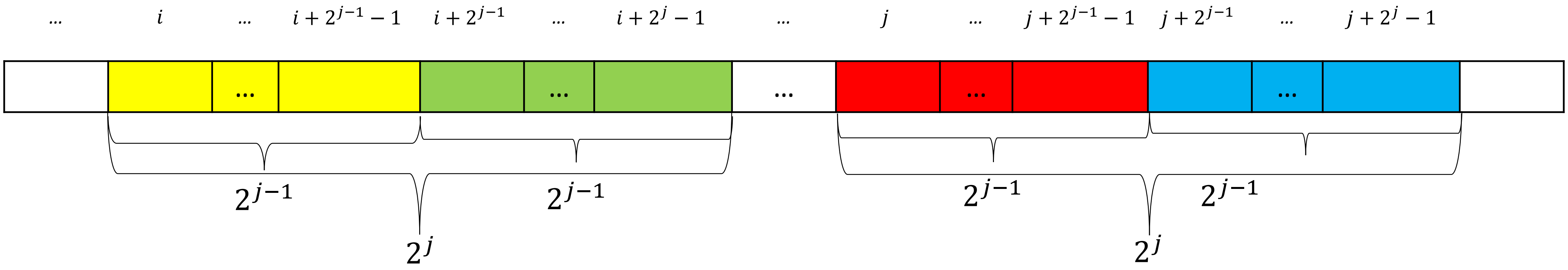
i	0	1	2	3	4	5	6	7
T	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$\boldsymbol{c}$	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$\$$
P^0	7	0	2	4	6	1	5	3
C^0	1	2	1	3	1	2	1	0

на 0-й фазе, когда упорядочиваются подстроки длины 1, сформировать указанные массивы можно устойчивым алгоритмом **сортировки подсчётом** за время $O(l + |\Sigma|)$

На фазах, начиная с 1-й, необходимо лексикографически упорядочивать подстроки длины которых больше 1.

Делать это можно, например, сортировкой слиянием, но надо научиться, используя информацию, полученную на предыдущих фазах, сравнивать две циклические подстроки одинаковой длины за константное время.

Сравнение двух циклических подстрок на j-ой фазе можно **выполнить за константное время,** так как мы знаем для каждой циклической подстроки длины 2^{j-1} к какому классу эквивалентности она отнесена, а любая циклическая подстрока длины 2^j состоит из двух циклических подстрок длины 2^{j-1}.



если $c^{j-1}[i] < c^{j-1}[j]$, то $t[i..i+2^j-1] < t[j..j+2^j-1]$

если $c^{j-1}[i] > c^{j-1}[j]$, то $t[i..i+2^j-1] > t[j..j+2^j-1]$

если $c^{j-1}[i] = c^{j-1}[j]$, то

если $c^{j-1}[i+2^{j-1}] < c^{j-1}[j+2^{j-1}]$, то $t[i..i+2^j-1] < t[j..j+2^j-1]$

иначе, если $c^{j-1}[i+2^{j-1}] > c^{j-1}[j+2^{j-1}]$, то $t[i..i+2^j-1] > t[j..j+2^j-1]$

иначе $t[i..i+2^j-1] = t[j..j+2^j-1]$

Массив $C^{\boldsymbol{j}}$ формируется на j –ой фазе за время $\Theta(l)$ после формирования массива $\boldsymbol{P^j}$, проходом по массиву $\boldsymbol{P^j}$ слева направо и сравнением (за константу) двух циклических подстрок длины 2^j на равенство:

- номер класса для $t\left[p^j[0]..p^j[0]+2^j-1\right]$ полагаем равным 0, т.е. $\mathrm{c}\left[p^j[0]\right] := 0$;

- для $1 \le i \le l$
  если $t\left[p^j[i]..p^j[i]+2^j-1\right] = t\left[p^j[i-1]..p^j[i-1]+2^j-1\right]$, то номер класса $c^j\left[p^j[i]\right]$ для $t\left[p^j[i]..p^j[i]+2^k-1\right]$ остается таким же, как и для $t\left[p^j[i-1]..p^j[i-1]+2^k-1\right]$ (т. е. $c^j\left[p^j[i]\right] = c^j\left[p^j[i-1]\right]$), иначе он увеличивается на 1 (т. е. $c^j\left[p^j[i]\right] := c^j\left[p^j[i-1]\right]+1$).

Так как к строке T добавляется \$, то это гарантирует, что каждый суффикс будет иметь свой класс эквивалентности.

2^0

i	0	1	2	3	4	5	6	7
T	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$\boldsymbol{c}$	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	\$
C^0	1	2	1	3	1	2	1	0

$l = 8$

2^1

i	0	1	2	3	4	5	6	7
T	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$\boldsymbol{c}$	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	\$
P^1	7	6	0	4	2	1	5	3
C^1	2	4	3	5	2	4	1	0

$t[i..i + 2^1 - 1]$

0	$a_{1кл}b_{2кл}$
1	$b_{2кл}a_{1кл}$
2	$a_{1кл}c_{3кл}$
3	$c_{3кл}a_{1кл}$
4	$a_{1кл}b_{2кл}$
5	$b_{2кл}a_{1кл}$
6	$a_{1кл}\ \$_{0кл}$
7	$\$_{0кл}a_{1кл}$

	P^1	$t[p^1[i]..p^1[i] + 2^j - 1]$
0	7	$\$_{0кл}a_{1кл}$
1	6	$a_{1кл}\ \$_{0кл}$
2	0	$a_{1кл}b_{2кл}$
3	4	$a_{1кл}b_{2кл}$
4	2	$a_{1кл}c_{3кл}$
5	1	$b_{2кл}a_{1кл}$
6	5	$b_{2кл}a_{1кл}$
7	3	$c_{3кл}a_{1кл}$

$\mathrm{c}[p[0]] = 0$

для $1 \le i \le l - 1$

если $t[p[i]..p[i] + 2^j - 1] = t[p[i-1]..p[i-1] + 2^j - 1]$,

то $\mathrm{c}[p[i]] = \mathrm{c}[p[i-1]]$, иначе $\mathrm{c}[p[i]] = \mathrm{c}[p[i-1]] + 1$).

2^1

i	0	1	2	3	4	5	6	7
T	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$\boldsymbol{c}$	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$\$$
C^1	2	4	3	5	2	4	1	0

2^2

i	0	1	2	3	4	5	6	7
T	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$\boldsymbol{c}$	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$\$$
P^2	7	6	4	0	2	5	1	3
C^2	3	6	4	7	2	5	1	0

	$t[i..i+2^2-1]$	P^2	$t[p^2[i]..p^2[i]+2^2-1]$
0	$(ab)_{2кл}(ac)_{3кл}$	7	$(\$a)_{0кл}(ba)_{4кл}$
1	$(ba)_{4кл}(ca)_{5кл}$	6	$(a\$)_{1кл}(ab)_{2кл}$
2	$(ac)_{3кл}(ab)_{2кл}$	4	$(ab)_{2кл}(a\$)_{1кл}$
3	$(ca)_{5кл}(ba)_{4кл}$	0	$(ab)_{2кл}(ac)_{3кл}$
4	$(ab)_{2кл}(a\$)_{1кл}$	2	$(ac)_{3кл}(ab)_{2кл}$
5	$(ba)_{4кл}(\$a)_{0кл}$	5	$(ba)_{4кл}(\$a)_{0кл}$
6	$(a\$)_{1кл}(ab)_{2кл}$	1	$(ba)_{4кл}(ca)_{5кл}$
7	$(\$a)_{0кл}(ba)_{4кл}$	3	$(ca)_{5кл}(ba)_{4кл}$

$\mathrm{c}[p[0]] = 0$

для $1 \le i \le l-1$

если $s[p[i]..p[i]+2^j-1] = s[p[i-1]..p[i-1]+2^j-1]$,

то $\mathrm{c}[p[i]] = \mathrm{c}[p[i-1]]$, иначе $\mathrm{c}[p[i]] = \mathrm{c}[p[i-1]]+1$).

2^2

i	0	1	2	3	4	5	6	7
T	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$\boldsymbol{c}$	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$
C^2	3	6	4	7	2	5	1	0

$l = 8$

2^3

i	0	1	2	3	4	5	6	7
T	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$\boldsymbol{c}$	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$
P^3	7	6	4	0	2	5	1	3
C^3	3	6	4	7	2	5	1	0

	$t[i..i+2^3-1]$		P^3	$t[p^3[i]..p^3[i]+2^3-1$
0	$(abac)_{3кл}(aba\$)_{2кл}$	0	7	$(\$aba)_{0кл}(caba)_{7кл}$
1	$(baca)_{4кл}(ba\$a)_{5кл}$	1	6	$(a\$ab)_{1кл}(acab)_{4кл}$
2	$(acab)_{4кл}(a\$ab)_{1кл}$	2	4	$(aba\$)_{2кл}(abac)_{3кл}$
3	$(caba)_{7кл}(\$aba)_{0кл}$	3	0	$(abac)_{3кл}(aba\$)_{2кл}$
4	$(aba\$)_{2кл}(abac)_{3кл}$	4	2	$(acab)_{4кл}(a\$ab)_{1кл}$
5	$(ba\$a)_{5кл}(baca)_{6кл}$	5	5	$(ba\$a)_{5кл}(baca)_{6кл}$
6	$(a\$ab)_{1кл}(acab)_{4кл}$	6	1	$(baca)_{4кл}(ba\$a)_{5кл}$
7	$(\$aba)_{0кл}(caba)_{7кл}$	7	3	$(caba)_{7кл}(\$aba)_{0кл}$

$\mathrm{c}[p[0]] = 0$
для $1 \le i \le l-1$
если $t[p[i]..p[i]+2^j-1] = t[p[i-1]..p[i-1]+2^j-1]$,
то $\mathrm{c}[p[i]] = \mathrm{c}[p[i-1]]$, иначе $\mathrm{c}[p[i]] = \mathrm{c}[p[i-1]]+1$).

После того, как выполнена сортировка циклических сдвигов строки $T + \$$, удалим из каждого циклического сдвига строки её суффикс, который начинается с $\$$, получим суффиксный массив для строки T:

i	0	1	2	3	4	5	6	7
T	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$\boldsymbol{c}$	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}$	$\$$
P^3	7	6	4	0	2	5	1	3
C^3	3	6	4	7	2	5	1	0

$l = 8$

0	7	$\cancel{\$abacaba}$
1	6	$a\cancel{\$abacab}$
2	4	$aba\cancel{\$abac}$
3	0	$abacaba\cancel{\$}$
4	2	$acaba\cancel{\$ab}$
5	5	$ba\cancel{\$abaca}$
6	1	$bacaba\cancel{\$a}$
7	3	$caba\cancel{\$aba}$

суффиксный массив для строки T $\longrightarrow$

0	7	–
1	6	a
2	4	aba
3	0	$abacaba$
4	2	$acaba$
5	5	ba
6	1	$bacaba$
7	3	$caba$

Если на j-й фазе выполнять лексикографическую сортировку l циклических подстрок длины 2^j $(\mathrm{j} < log_2|T| + 1)$ **сортировкой слиянием**, то так как сравнение двух циклических подстрок мы выполняем за $\mathrm{O}(1)$, а массив C формируется за $\mathrm{O}(|T|)$, то время выполнения одной фазы

$$\mathrm{O}(|T| \cdot \log|T|) + \mathrm{O}(|T|) = \mathrm{O}(|T| \cdot \log|T|).$$

Учитывая, что число фаз $k = \lceil \log|T| \rceil < \log|T|+1$, получаем, что **время работы алгоритма лексикографической сортировки циклических сдвигов строки** (построения суффиксного массива строки T)

$$\boldsymbol{O}(|T| \cdot \mathbf{log^2}|T|).$$

Требуемая **память** $- \boldsymbol{O}(|T|)$.

Задан фиксированный текст $\mathrm{T} = (t_0, t_1, \ldots, t_{l-1})$	**On-line поступают образцы из множества** $S = \{S_0, S_1, \ldots, S_{n-1}\}$	**Определить, встречается ли образец из S в качестве подстроки в тексте** T

Структура данных	Время построения	Память	Время поиска
Суффиксный массив	$\mathrm{O}(\lvert T\rvert \cdot log^2\lvert T\rvert)$	$\mathrm{O}(\lvert T\rvert)$	$\mathrm{O}\left(log\lvert T\rvert \cdot \sum_{i=1}^{n-1} \lvert S_i\rvert\right)$
Суффиксный бор	$\mathrm{O}(\lvert T\rvert^2)$	$\mathrm{O}(\lvert T\rvert^2 \cdot \lvert\Sigma\rvert)$	$\mathrm{O}\left(\sum_{i=0}^{n-1} \lvert S_i\rvert\right)$

Спасибо за внимание!