

# Алгоритм Карпа-Рабина

$ns : \Sigma \rightarrow [0.. |\Sigma| - 1]$  – порядок символов в  $\Sigma$ .  $s = |\Sigma|$ .

$H(P) = ns(p_1) \times s^{m-1} + ns(p_2) \times s^{m-2} \dots ns(p_{m-1}) \times s + ns(p_m)$  и

$H(T[i : i + m - 1]) = ns(t_i) \times s^{m-1} + ns(t_{i+1}) \times s^{m-2} \dots ns(t_{i+m-2}) \times s + ns(t_{i+m-1})$ .

Если  $H(P) = H(T[i : i + m - 1])$ , то образец встретился в  $i$ -й поз. текста.

## Рекуррентное хеширование:

$H(T[i + 1 : i + m]) = (H(T[i : i + m - 1]) - ns(t_i) \times s^{m-1}) \times s + ns(t_{i+m})$ .

**Пример.**  $\Sigma = \{acgt\}$ ,  $P = acat$ ,  $T = ggacataccagac$ ;

$ns(a) = 0$ ;  $ns(c) = 1$ ;  $ns(g) = 2$ ;  $ns(t) = 3$ ;

$$H(P) = 0 \times 4^3 + 1 \times 4^2 + 0 \times 4^1 + 3 = 19;$$

$$H(T[1 : 4]) = 2 \times 4^3 + 2 \times 4^2 + 0 \times 4^1 + 1 = 161;$$

$$H(T[2 : 5]) = 2 \times 4^3 + 0 \times 4^2 + 1 \times 4^1 + 0 = 132 = (161 - 2 \times 4^3) \times 4 + 0;$$

$$H(T[3 : 6]) = 0 \times 4^3 + 1 \times 4^2 + 0 \times 4^1 + 3 = 19 = (132 - 2 \times 4^3) \times 4 + 3;$$

## Обобщения задачи поиска образца:

- Образцы с джокерами:  $x$  – любой символ  
Пример.  $P = abxhcx$  содержится в тексте  $g**ab**d**cc**bab**ab**ca**d**$  дважды.
- Образцы, позиции которых заданы множествами символов  
a- [a,b]-c-[c,d]-[a,c,d]-[c,d]-a  
a- [a,b]-c-[c,d]-  $\neg$ b - x - a
- Поиск образца с допустимым уровнем искажений:  
abcdab – ab**d**dab – abcd**c**b – ab**cc**dab – abcdab
- Поиск множества образцов
- Комбинации задач (например, поиск множества образцов, позиции которых заданы множествами символов)
- Образцы с переменными.  $X \in \Sigma^*$ .  
 $P = abXXcX$ :  $abccsc$ ;  $ab$  $abb$  $abb$  $cab$  $b$
- Параметризованные образцы. 2 алфавита:  $\Sigma$  и  $\Pi$ :  
Образцы  $\pi$ -согласованы, если  $\exists f: \Pi \rightarrow \Pi$   
 $\Sigma = \{a,b,c\}$ ;  $\Pi = \{X, Y, Z, T\}$   
 $abcXbbYYccZ$  и  $abcZbbXXccT$

## Алгоритм Ахо-Корасик

**Задача.** Задано множество образцов  $P = \{P_1, P_2, \dots, P_z\}$ . Требуется обнаружить все вхождения в текст  $T$  любого образца из  $P$ .

$i$ -й образец  $P_i = p_{i1}p_{i2}\dots p_{i,m_i}$  имеет длину  $m_i$ ;  $p_{i,j} \in \Sigma$ .

Текст  $T = t_1 t_2 \dots t_N$ ,  $t_k \in \Sigma$ ,  $1 \leq k \leq N$ .

Это обобщение называют множественной задачей точного поиска или задачей поиска по групповому запросу

Наивный алгоритм решает эту задачу путем поиска каждого образца из набора с использованием любого из рассмотренных выше линейных алгоритмов. Такой поиск имеет трудоемкость  $O(zN + \sum_i m_i)$ .

Эффективный алгоритм решения этой задачи имеет трудоемкость  $O(N + \sum_i m_i)$ .

# Алгоритм Ахо-Корасик

- Этап предобработки: построение ДКА по исходному множеству образцов
- Этап поиска: однократный "прогон" текста через этот автомат.

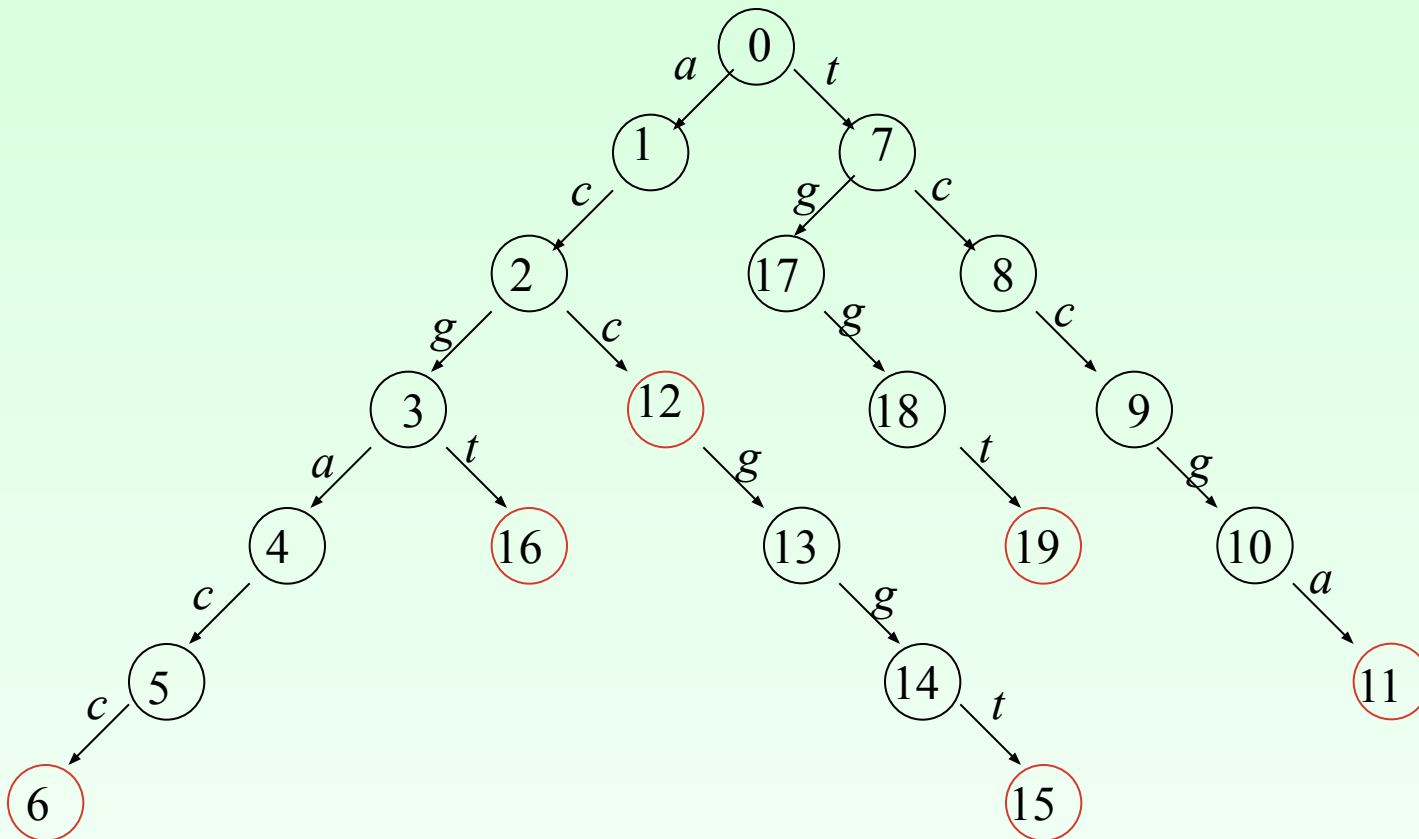
## 1. Этап предобработки.

Сначала строится "машина идентификации цепочек"  $M_p$ . Работа машины  $M_p$  описывается тремя функциями: функцией переходов  $\varphi(s, a)$  ( $s$  – состояние машины,  $a \in \Sigma$ ), функцией отказов  $f(s)$  и функцией выходов  $o(s)$ .

# Алгоритм Ахо-Корасик

Функция переходов  $\varphi(s,a)=s'$ , если существует выходящее из  $s$  ребро, помеченное символом " $a$ " и связывающее состояния  $s$  и  $s'$ ; в противном случае  $\varphi(s,a) = \langle \text{fail} \rangle = -1$

**Пример.** Пусть  $\Sigma = \{a,c,g,t\}$ ;  $P = \{acgacc, tcgga, accggt, acgt, acc, tggt\}$ ;  
 $\varphi(7, g) = 17$ ;  $\varphi(3, a) = 4$ ;



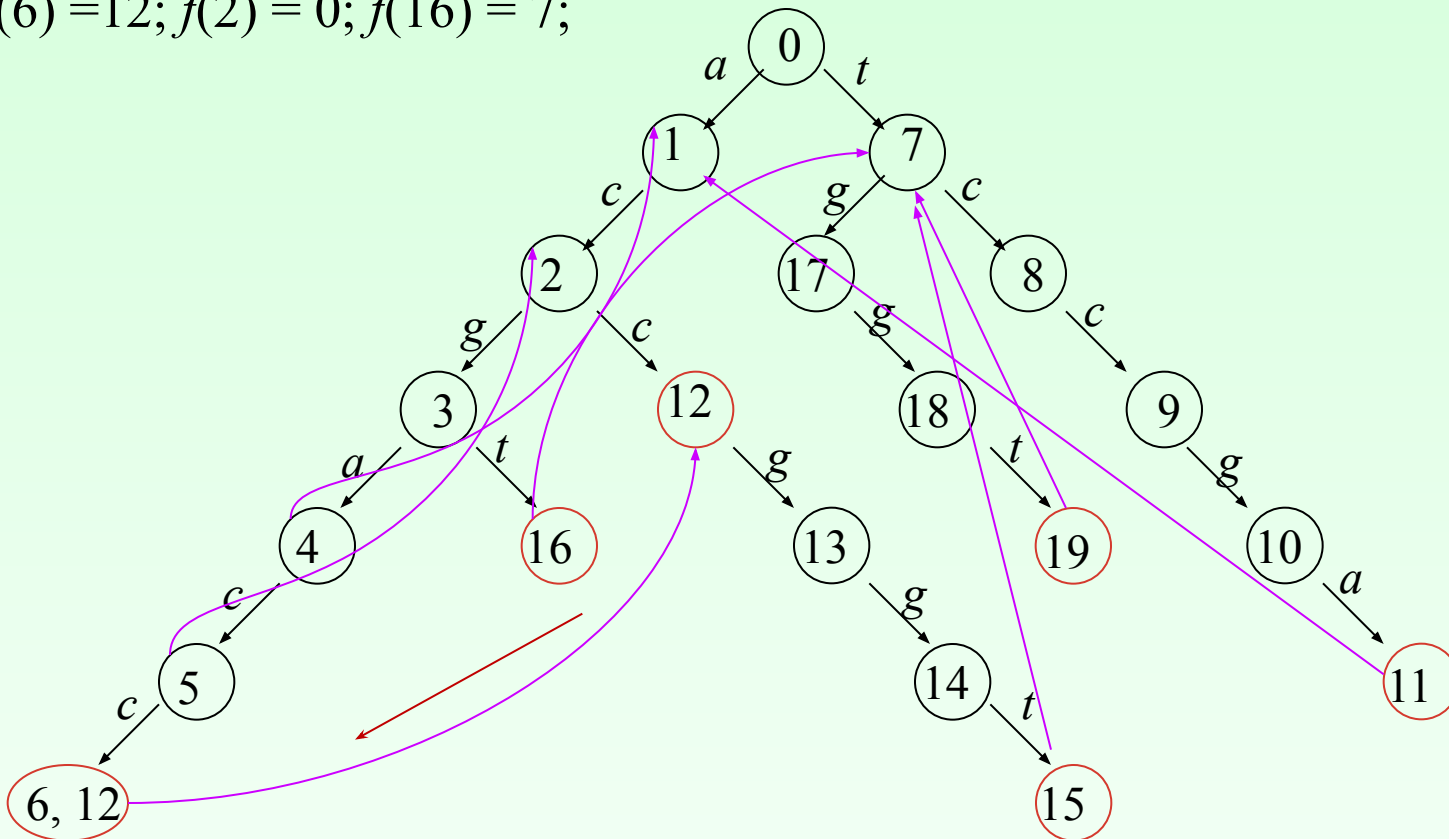
# Алгоритм Ахо-Корасик.

Построение  $f(s)$ : пусть  $\varphi(s\_pred, a) = s, f(s\_pred) = s''$ .

Метка : Если  $\varphi(s'', a) \neq fail$ , то  $f(s) = \varphi(s'', a)$ ;  $o(s) := o(s) \cup o(f(s))$ ,  
иначе  $s'' := f(s'')$ ; goto Метка.

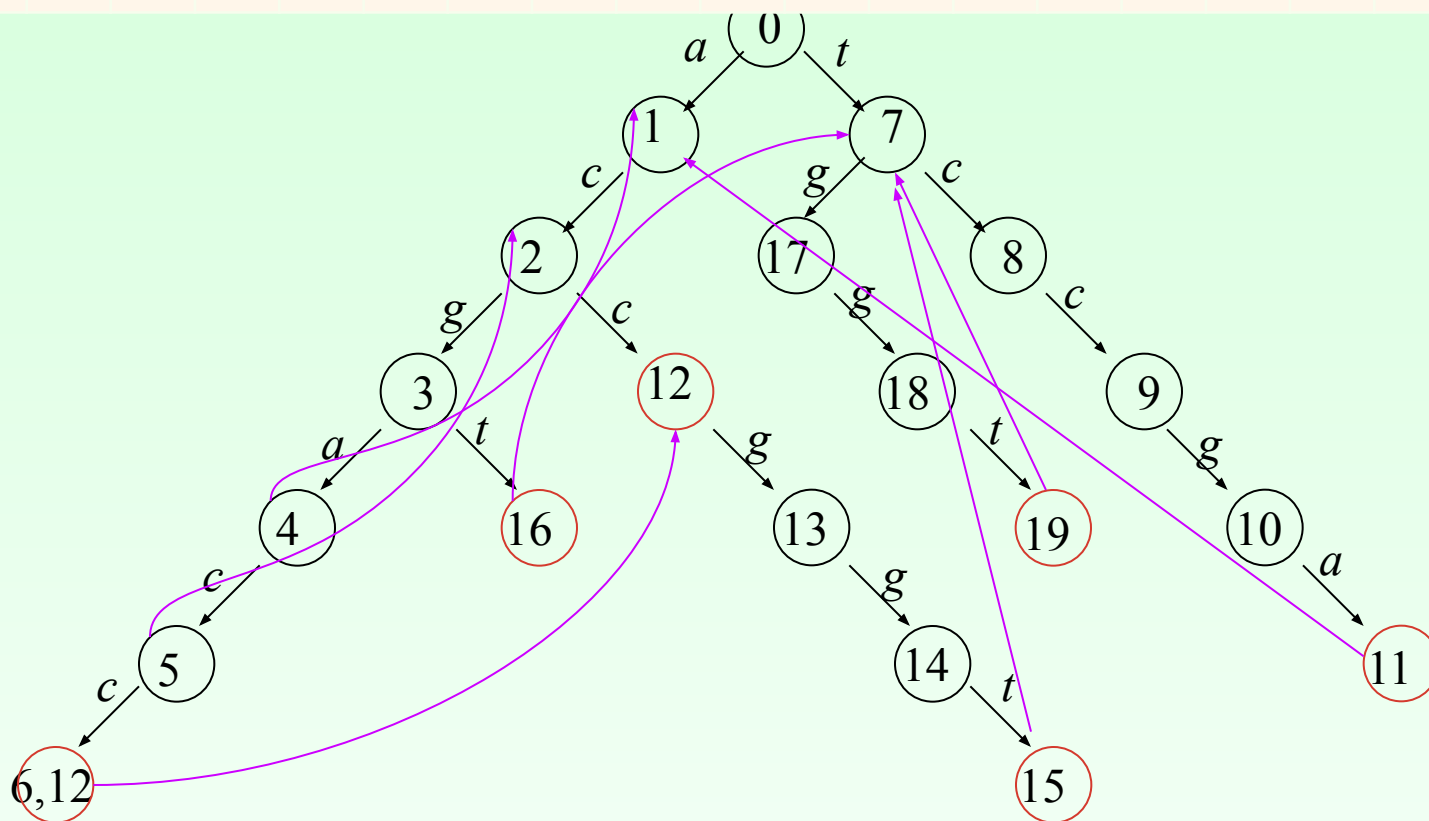
Порядок построения: по уровням дерева (структура «очередь»).

**Пример.** Пусть  $\Sigma = \{a, c, g, t\}$ ;  $P = \{acgatc, tcsga, accggt, acgt, acc, tggt\}$ ;  $o(6) = \{1, 4\}$ ;  
 $f(6) = 12$ ;  $f(2) = 0$ ;  $f(16) = 7$ ;



# Алгоритм Ахо-Корасик.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	1	1	1	4	1	1	1	1	1	1	11	1	...							1
C	0	2	0	0	5	6	0	8	9	0	0	2								8
G	0	0	3	0	0	3	13	17	0	10	0	0								17
T	7	7	7	16	7	7	7	7	7	7	7	7								7



**Повтор** — пара совпадающих фрагментов текста.

## Классификация повторов

• По способу их прочтения и использованию переименований:

**Повтор** (в широком смысле) — пара фрагментов текста, совпадающих с точностью до переименования элементов алфавита и (или) изменения направления считывания.

Типы повторов:

- прямые :  $\dots \text{AGTTC} \dots \text{AGTTC} \dots$
- симметричные:  $\dots \overleftarrow{\text{AGTTC}} \dots \overrightarrow{\text{CTTGA}} \dots$
- с точностью до подстановки на элементах алфавита: секвентные переносы в музыке; замены  $0 \leftrightarrow 1$ ;  $A \leftrightarrow T$ ,  $C \leftrightarrow G$ .
  - прямые комплементарные:  $\dots \overrightarrow{\text{AGTTC}} \dots \overleftarrow{\text{TCAAG}} \dots$
  - инвертированные:  $\dots \text{AGTTC} \dots \overleftarrow{\text{GAAC}} \dots$



# Повторы в текстах

Слова и словосочетания.

Повтор предложения (абзаца) – признак ошибки.

Мелодические обороты.

## **В ДНК-последовательностях:**

1. Участки с аномальным нуклеотидным составом: (А,Т)-богатые...
2. Участки микросателлитной ДНК: периодичности с малой длиной периода (2-3) и достаточно высокой кратностью повторений.
3. Тандемные повторы с произвольной длиной периода.
4. Разнесенные повторы значительной длины.
5. Мобильные элементы

# Классификация повторов

- По наличию искажений:

Повторы могут быть

совершенные:

... AGTTC ... AGTTC...

и несовершенные (с заменами, вставками, делециями):

... A**G**TTC ... A**A**TTC ... (замена),

... AGTTC ... AGT**T**TTC... (вставка),

в том числе с точностью до агрегирования:

... AGTTC ... **GATCT** ...

(совпадают при заменах  $\{A,G\} \rightarrow P_u$ ,  $\{C,T\} \rightarrow P_u$ ).

# Классификация повторов

- По характеру расположения в тексте:

будем различать повторы

разнесенные

... AGTTC ... AGTTC...

тандемные

... AGTTC AGTTC...

с наложением :

... AGTTCAGTTCAGTTC ...  
AGTTC

## Представление текста в терминах повторов

Полный частотный спектр текста.

Пусть

$\Sigma$  – конечный алфавит;


$S$  – текст, составленный из элементов  $\Sigma$ ;

$N = |S|$  – длина текста;

$S[i] = s_i$  –  $i$ -й элемент текста  $S$  ( $1 \leq i \leq N$ );

$S[i:j]$  – фрагмент текста, включающий элементы с  $i$ -го по  $j$ -й ( $1 \leq i < j \leq N$ ).

$l$ -грамма – связная цепочка текста длины  $l$  ( $S[i:i+l-1]$ ).

$$S = s_1 s_2 s_3 s_4 s_5 \cdots s_N$$


Полное число  $l$ -грамм:  $N - l + 1$ .

Число различных  $l$ -грамм:  $M_l \leq N - l + 1$ .

**Частотная характеристика  $l$ -го порядка** текста  $S$  – совокупность элементов  $\Phi_l(S) = \{\varphi_{l1}, \varphi_{l2}, \dots, \varphi_{lM_l}\}$  где  $\varphi_{li}$  ( $1 \leq i \leq M_l$ ) есть пара : «  $i$ -я  $l$ -грамма –  $x_i$ , ее частота в тексте –  $F_l(x_i)$  ».

$l_{\max}(S)$  – наибольшее  $l$ , при котором в  $S$  есть *повторяющиеся*  $l$ -граммы.

Совокупность частотных характеристик

$$\Phi(S) = \{\Phi_1(S), \Phi_2(S), \dots, \Phi_{l_{\max}+2}(S)\}$$

называется *полным частотным спектром текста  $S$* .

**Усеченный спектр**

$$\Phi^*(S) = \{\Phi^*_1(S), \Phi^*_2(S), \dots, \Phi^*_{l_{\max}}(S)\}$$

$\Phi^*_l(S)$  отличается от  $\Phi_l(S)$  отсутствием  $l$ -грамм с единичной частотой встречаемости.

**Пример.** Пусть  $S = \text{саабсabbса}$ .

Тогда  $\Phi^*(S) = \{\Phi^*_1(S), \Phi^*_2(S), \Phi^*_3(S)\}$ , где

$$\Phi^*_1(S) = \{\langle a, F_1(a) = 4 \rangle; \quad \langle b, F_1(b) = 3 \rangle; \quad \langle c, F_1(c) = 3 \rangle\};$$

$$\Phi^*_2(S) = \{\langle ca, F_2(ca) = 3 \rangle; \quad \langle ab, F_2(ab) = 2 \rangle; \quad \langle bc, F_2(bc) = 2 \rangle\};$$

$$\Phi^*_3(S) = \{\langle bса, F_3(bса) = 2 \rangle\};$$

Для повторов значительной длины спектр можно дополнить позиционной информацией.

## Наиболее важными являются следующие параметры частотного спектра:

$l_{\max}$  — **длина максимального повтора в тексте.**

Для случайного текста длины  $N$  с вероятностями появления элементов алфавита  $p_r$  ( $1 \leq r \leq n = |\Sigma|$ ) можно пользоваться оценкой: .

Если реальная длина  $l_{\max}$  в тексте существенно превышает ожидаемое значение, это свидетельствует о наличии дубликативных механизмов порождения текста.

$$l_{\max} \sim 2 \ln N / \left| \ln \sum_{r=1}^n p_r^2 \right|$$

$M_l$  — **размер словаря  $l$ -грамм.**

Он фигурирует в определении комбинаторной сложности текста.

— **максимальное значение частот** встречаемости

$$F_l^{\max} = \max_{1 \leq i \leq M_l} F_l(x_i) \quad l\text{-грамм в тексте } (1 \leq l \leq l_{\max});$$

— **минимальное значение частот** встречаемости

$$F_l^{\min} = \min_{1 \leq i \leq M_l} F_l(x_i) \quad l\text{-грамм в тексте } (1 \leq l \leq l_{\max}); \text{ представляет интерес лишь при малых значениях } l; \text{ при больших, обычно } F_l^{\min} = 1.$$

$E_l^k$  — **Число** различных  $l$ -грамм, каждая из которых встречается в тексте ровно  $k$  раз ( $k = 1, 2, \dots, N - l + 1$ ); зависимость  $E_l^k$  от  $k$  при фиксированном  $l$  является аналогом известной в лингвистике кривой Юла;

$M_l - E_l^1$  — **число** различных повторяющихся  $l$ -грамм;

$E_l^0$  — число  $l$ -грамм, ни разу **не встретившихся** в тексте.

Наличие таковых в ситуации, когда  $N / n^l \gg 1$ , можно трактовать как аномальный эффект.

Имеют место простые соотношения, связывающие основные параметры:

$$M_l = \sum_{k=1}^{F_l^{\max}} E_l^k, \quad N - l + 1 = \sum_{k=1}^{F_l^{\max}} k \cdot E_l^k, \quad E_l^0 = n^l - M_l.$$



# Алгоритмы отыскания совершенных повторов

## Метод лексикографической сортировки

Лексикографический порядок слов  $u = u_1 \dots u_p$  и  $v = v_1 \dots v_q$  :  
 $u \leq v$ , если  $u = v$  или  $(\exists j$ , такое что  $u_j < v_j$  и  $u_i = v_i \ \forall i < j)$  или  $(p < q$  и для всех  $i \leq p \ u_i = v_i)$ .

**Пример.**  $S = \text{abcdabcbscbabcd}$ ;  $l = 3$ ;

abc  $f(\text{abc}) = 3$       аналог для произвольного  $l$ -

abc                      суффиксный массив

abc

bab

bcb  $f(\text{bcb}) = 2$

bcb

bcd  $f(\text{bcd}) = 2$

bcd

cba

cbc

cda

dab

# Числовая сортировка

## Задача:

Дан *массив*  $A$  из  $n$  *элементов*:  $a_1, a_2, \dots, a_n$

С каждым элементом  $a_i$  связан *ключ*  $k_i \in K$ .

На множестве ключей  $K$  задано отношение порядка — **линейного (или совершенного) упорядочивания**, для которого выполнены следующие условия:

*закон трихотомии*: для любых  $x, y \in K$  либо  $x < y$ , либо  $x > y$ , либо  $x = y$ ;

*транзитивность*: для любых  $x, y, z \in K$  если  $x < y$  и  $y < z$ , то  $x < z$ .

**Задачей сортировки по неубыванию** является нахождение перестановки элементов  $p(1), p(2), \dots, p(n)$  при которой ключи располагаются в порядке неубывания:  $k_{p(1)} \leq k_{p(2)} \leq \dots \leq k_{p(n)}$ .

В результате работы алгоритма и применения перестановки получается отсортированный массив:  $a_{p(1)}, a_{p(2)}, \dots, a_{p(n)}$

Аналогично можно определить **сортировку по невозрастанию**.

# Числовая сортировка

Сортировка выбором (*Selection sort*) :

- находим номер минимального значения в текущем списке
- производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
- сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

**5 9 4 7 2 4 1 6**

1 **9** 4 7 **2** 4 5 6

1 2 **4** 7 9 4 5 6

1 2 4 **7** 9 4 5 6

1 2 4 4 **9** 7 **5** 6

1 2 4 4 5 **7** 9 **6**

1 2 4 4 5 6 **9** **7**

1 2 4 4 5 6 7 **9**

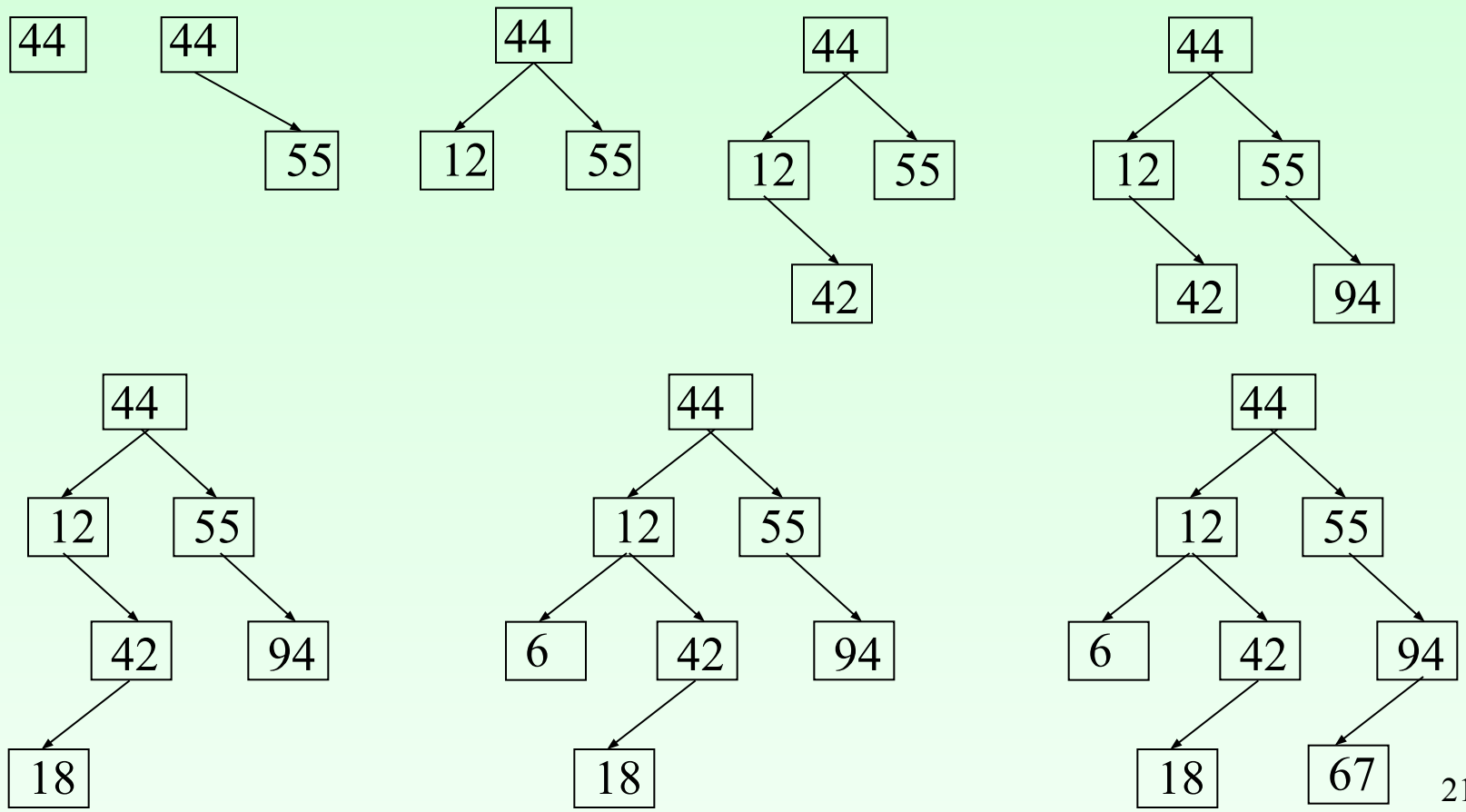
# Числовая сортировка

## •Сортировка пузырьком (сортировка всплыванием)

•Элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает»).

•Первый цикл:	второй цикл	третий цикл	четвертый цикл
• <b>5</b> <u>9</u> 4 7 2 4 1 6	<b>5</b> 4 7 2 4 1 6 9	4 <b>5</b> <u>2</u> 4 1 6 7 9	<u>4</u> 2 4 1 5 6 7 9
•5 4 <b>9</b> <u>7</u> 2 4 1 6	4 <b>5</b> <u>7</u> 2 4 1 6 9	4 2 <b>5</b> <u>4</u> 1 6 7 9	<b>2</b> <u>4</u> 4 1 5 6 7 9
•5 4 7 <b>9</b> <u>2</u> 4 1 6	4 <b>5</b> <u>7</u> 2 4 1 6 9	4 2 4 <b>5</b> <u>1</u> 6 7 9	2 4 1 <b>4</b> <u>5</u> 6 7 9
•5 4 7 2 <b>9</b> <u>4</u> 1 6	4 5 <b>2</b> <u>7</u> 4 1 6 9	4 2 4 1 <b>5</b> <u>6</u> 7 9	пятой цикл
•5 4 7 2 4 <b>9</b> <u>1</u> 6	4 5 2 4 <b>7</b> <u>1</u> 6 9	2 1 4 4 5 6 7 9	
•5 4 7 2 4 1 <b>9</b> <u>6</u>	4 5 2 4 1 <b>7</b> <u>6</u> 9	шестой цикл	
•5 4 7 2 4 1 6 <b>9</b>	4 5 2 4 1 6 <b>7</b> <u>9</u>	1 2 4 4 5 6 7 9	

**Сортировка деревом.** При добавлении в дерево нового элемента его последовательно сравнивают с нижестоящими узлами. Если элемент  $\geq$  корня - он идет в правое поддерево, сравниваем его уже с правым сыном, иначе - он идет в левое поддерево, сравниваем с левым, и так далее, пока есть сыновья, с которыми можно сравнить. **44 55 12 42 94 18 06 67**



# Числовая сортировка

## Сортировка вычерпыванием :

Пусть известно, что максимальный элемент сортируемого массива не превосходит некоторое натуральное  $m$ :

- Организовать  $m$  пустых очередей по одной для каждого натурального числа от 1 до  $m$ . Каждую такую очередь называют *черпаком*.
- Просмотреть последовательность  $A$  слева направо, помещая элемент  $a_i$  в очередь с номером  $a_i$ .
- Сцепить эти очереди, т.е. содержимое  $(i+1)$ -й очереди приписать к концу  $i$ -й очереди ( $i=1,2,\dots, m-1$ ), получив тем самым упорядоченную последовательность.

**Пример:** 1 3 2 5 2 5 1 2 1 5 4 3 4 5. Просмотрели 5 элементов

1: 1

2: 2 2

3: 3

4:

5: 5

# Числовая сортировка

## Сортировка вычерпыванием :

Пусть известно, что максимальный элемент сортируемого массива не превосходит некоторое натуральное  $m$ :

- Организовать  $m$  пустых очередей по одной для каждого натурального числа от 1 до  $m$ . Каждую такую очередь называют *черпаком*.
- Просмотреть последовательность  $A$  слева направо, помещая элемент  $a_i$  в очередь с номером  $a_i$ .
- Сцепить эти очереди, т.е. содержимое  $(i+1)$ -й очереди приписать к концу  $i$ -й очереди ( $i=1,2,\dots, m-1$ ), получив тем самым упорядоченную последовательность.

**Пример:** 1 3 2 5 2 5 1 2 1 5 4 3 4 5 : 1 1 1 2 2 2 3 3 4 4 5 5 5 5

1: 1 1 1

2: 2 2 2

3: 3 3

4: 4 4

5: 5 5 5 5

# Лексикографическая сортировка на основе сортировки вычерпыванием :

- $l$ -граммы сортируем по позиции  $l$
- Полученный список сортируем по позиции  $l - 1$
- ...
- Полученный список сортируем по позиции 1

Пример: abcdabcbscbabcd;  $l = 3$ ;

- 1.abc
- 2.bcd
- 3.cda
- 4.dab
- 5.abc
- 6.bcb
- 7.cbc
- 8.bcb
- 9.cba
- 10.bab
- 11.abc
- 12.bcd



# Лексикографическая сортировка на основе сортировки вычерпыванием

- $l$ -граммы сортируем по позиции  $l$
- Полученный список сортируем по позиции  $l - 1$
- ...
- Полученный список сортируем по позиции 1

**Пример:** `abcdabcbscbabcd`;  $l = 3$ ;  
`abc, bcd, cda, dab, abc, bcb, cbc, bcb,`  
`cba, bab, abc, bcd;`

## Сортировка по 3-й позиции:

a: `cda, cba`  
b: `dab, bcb, bcb, bab`  
c: `abc, abc, cbc, abc`  
d: `bcd, bcd`

Список  $l$ -грамм после первого этапа:  
`cda, cba, dab, bcb, bcb, bab, abc, abc,`  
`cbc, abc, bcd, bcd`

## Сортировка по 2-й позиции:

a: `dab, bab`  
b: `cba, abc, abc, cbc, abc`  
c: `bcb, bcb, bcd, bcd`  
d: `cda`

Список  $l$ -грамм после второго этапа:  
`dab, bab, cba, abc, abc, cbc, abc, bcb,`  
`bcb, bcd, bcd, cda`

## Сортировка по 1-й позиции:

a: `abc, abc, abc`  
b: `bab, bcb, bcb, bcd, bcd`  
c: `cba, cbc, cda`  
d: `dab`

Список  $l$ -грамм после 3 этапа:  
`abc, abc, abc, bab, bcb, bcb, bcd, bcd,`  
`cba, cbc, cda, dab`

## Алгоритмы отыскания совершенных повторов

**Метод, основанный на хешировании.**

**Хеширование** – отображение, которое ставит в соответствие произвольной  $l$ -грамме текста  $x_i$  ( $1 \leq i \leq N - l + 1$ ) число  $h(x_i)$ , (адрес, по которому хранится информация об  $x_i$ ).

**Пример** простейшего отображения – представление  $l$ -граммы  $x_i = a_{i1}a_{i2} \dots a_{il}$ , где  $a_{ij} \in \Sigma$  ( $j = 1 \div l$ ) в  $q$ -ичной системе счисления, где  $q = |\Sigma|$ .

$$h_1(x_i) = k_1q^{l-1} + k_2q^{l-2} + \dots + k_lq^0 = \sum_{i=1}^l k_iq^{l-i}$$

( $0 \leq k_i \leq q - 1$ ).

**Недостаток** этого отображения – большой (порядка  $|\Sigma|^l$ ) диапазон изменения чисел  $h_1(x_i)$  (сильно разреженный массив адресов).

**Достоинство** – отображение  $h_1$  **взаимно-однозначное** и достаточно **просто вычислимо**.

Трудоёмкость «рекуррентного хеширования» —  $O(N)$ .

Компромисс по памяти и по времени может быть достигнут, если сузить диапазон изменения возможных значений  $h(x)$ .

Для этого:

- а) отказаться от требования однозначности функции расстановки  $h(x_i)$ ,
- в) разделить «наложившиеся» объекты с помощью специальной техники (открытая адресация, перехеширование, использование списковых структур и т. д.).

Пример функции расстановки, допускающей наложения:

$$h_2(x_i) = h_1(x_i) \bmod M$$

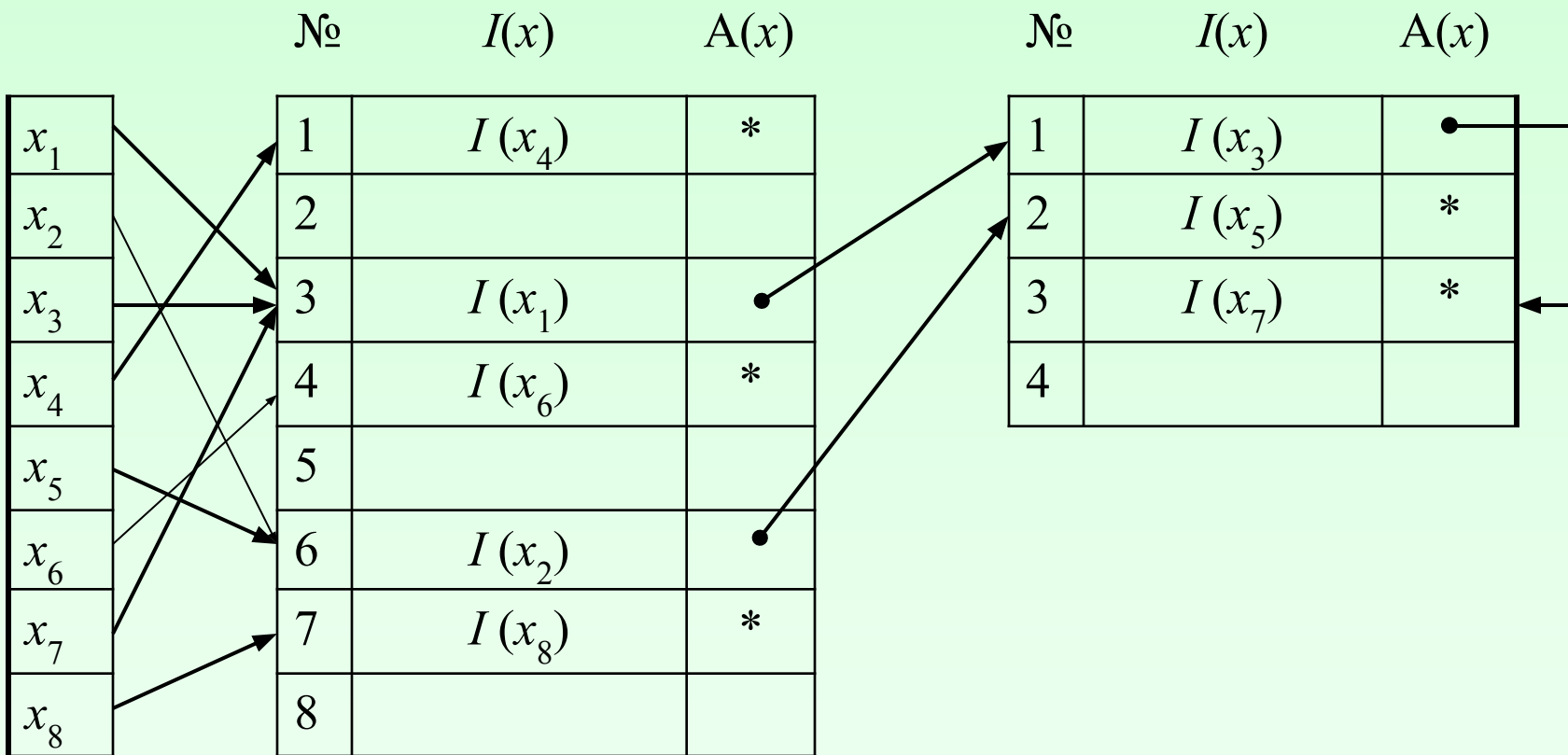
$M$  - простое число (размер поля).

## Пример списковой схемы устранения наложений

Информационный массив

Расстановочное поле

Дополнительное поле (ДП)



## Префиксное и суффиксное деревья

Если  $v = xyz$ , то  $x$  – префикс  $v$ ,  $z$  – суффикс,  $y$  – подслово.

Оптимальные алгоритмы отыскания совершенных повторов основаны на построении префиксного дерева, суффиксного дерева или графа подслов текста (DAtG).

Первая конструкция разработана Вайнером ( Weiner P., 1973), вторая – Мак-Крейгом (McCreight, 1976), третья – (A.Blumer, J.Blumer, A.Ehrenfeucht, et al., 1984). Все конструкции функционально эквивалентны и реализуются за линейное (в зависимости от длины текста) время с линейными затратами памяти.

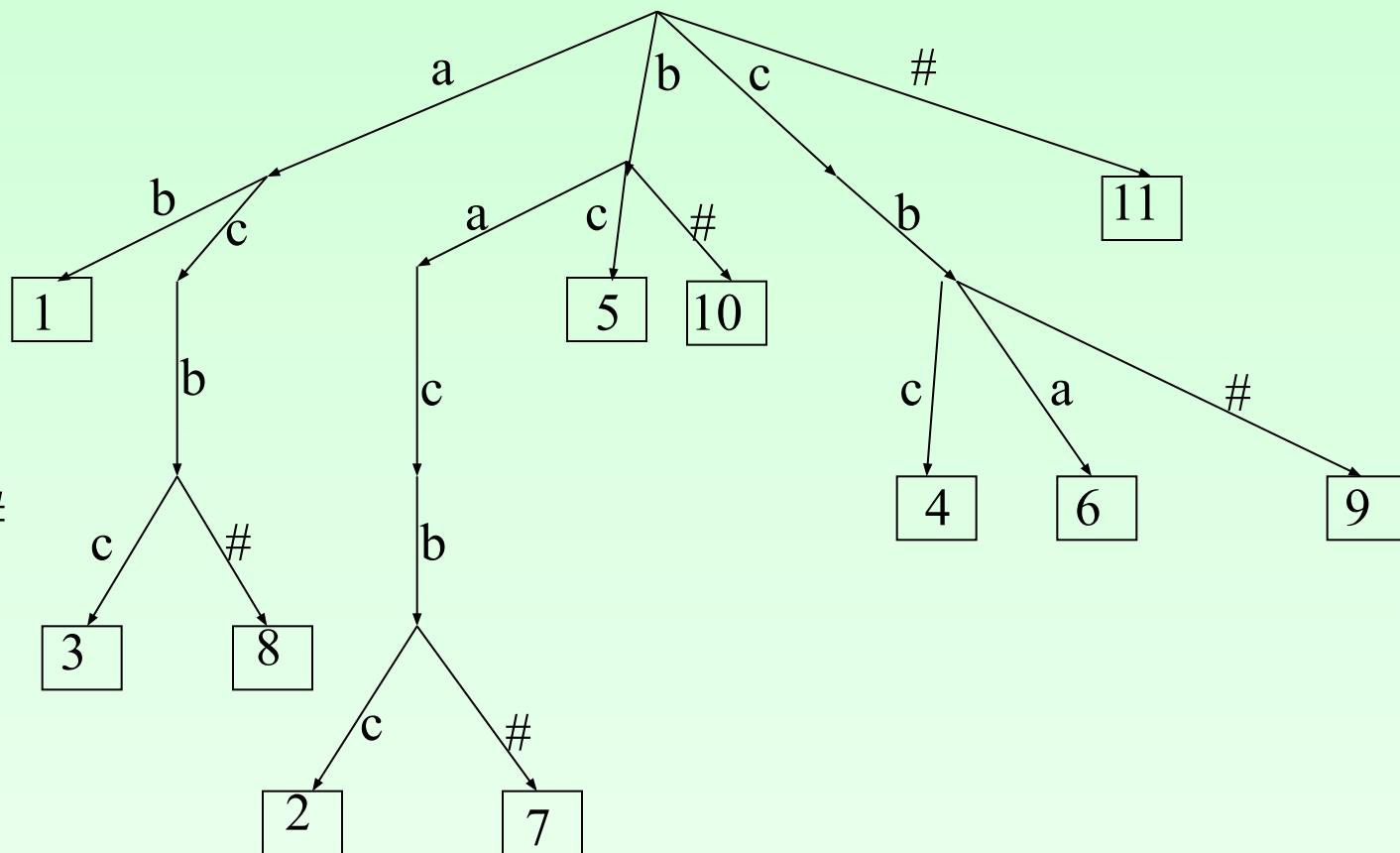
Префикс–идентификатор  $pr(i)$  позиции  $i$  в тексте  $T$  – кратчайшее подслово, начинающееся в позиции  $i$  и встречающееся в  $T \#$  только один раз ( $\#$  – конечный маркер).

Префиксное дерево = дерево префикс-идентификаторов.

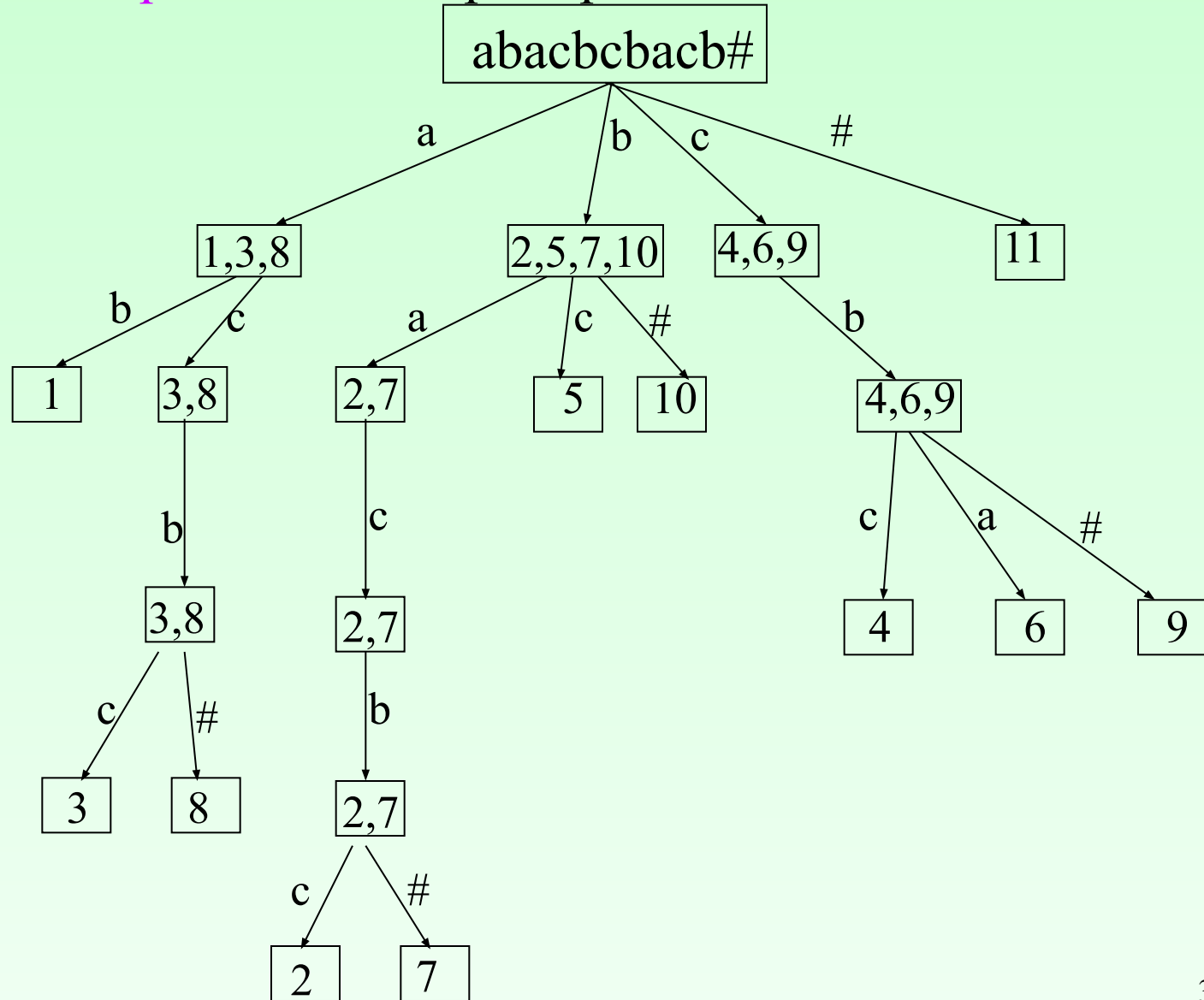
**Пример** дерева префикс-идентификаторов для  $T\# = abacbcabac\#$

$i$     $pr(i)$

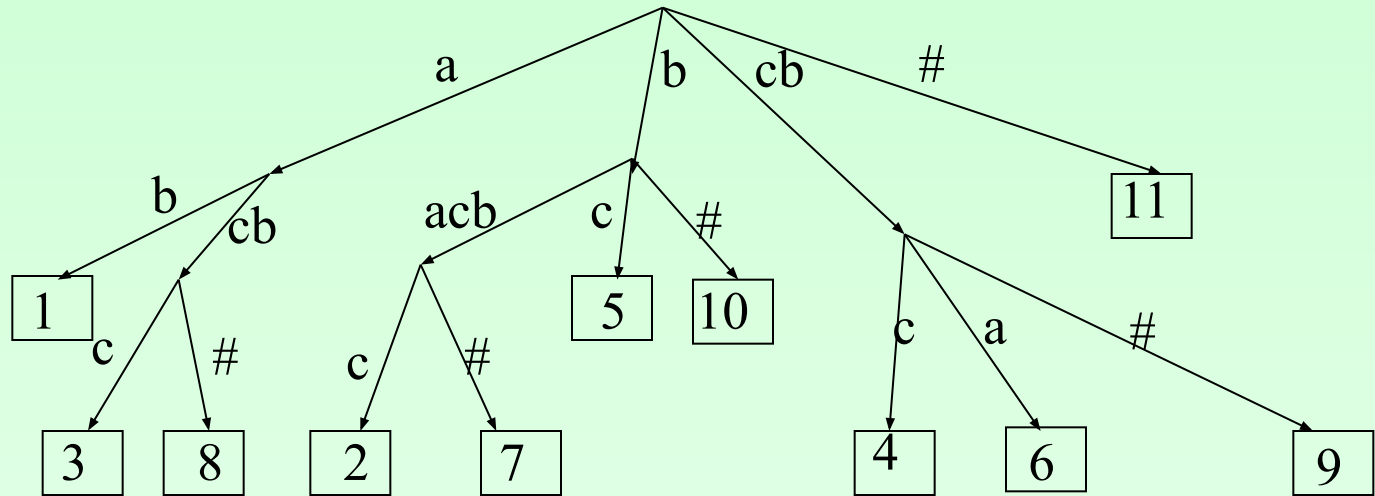
1. ab
2. bacbc
3. acbc
4. cbc
5. bc
6. cba
7. bacb#
8. acb#
9. cb#
10. b#
11. #



# Алгоритм Мартинеца на примере T# = abacbcbacb#

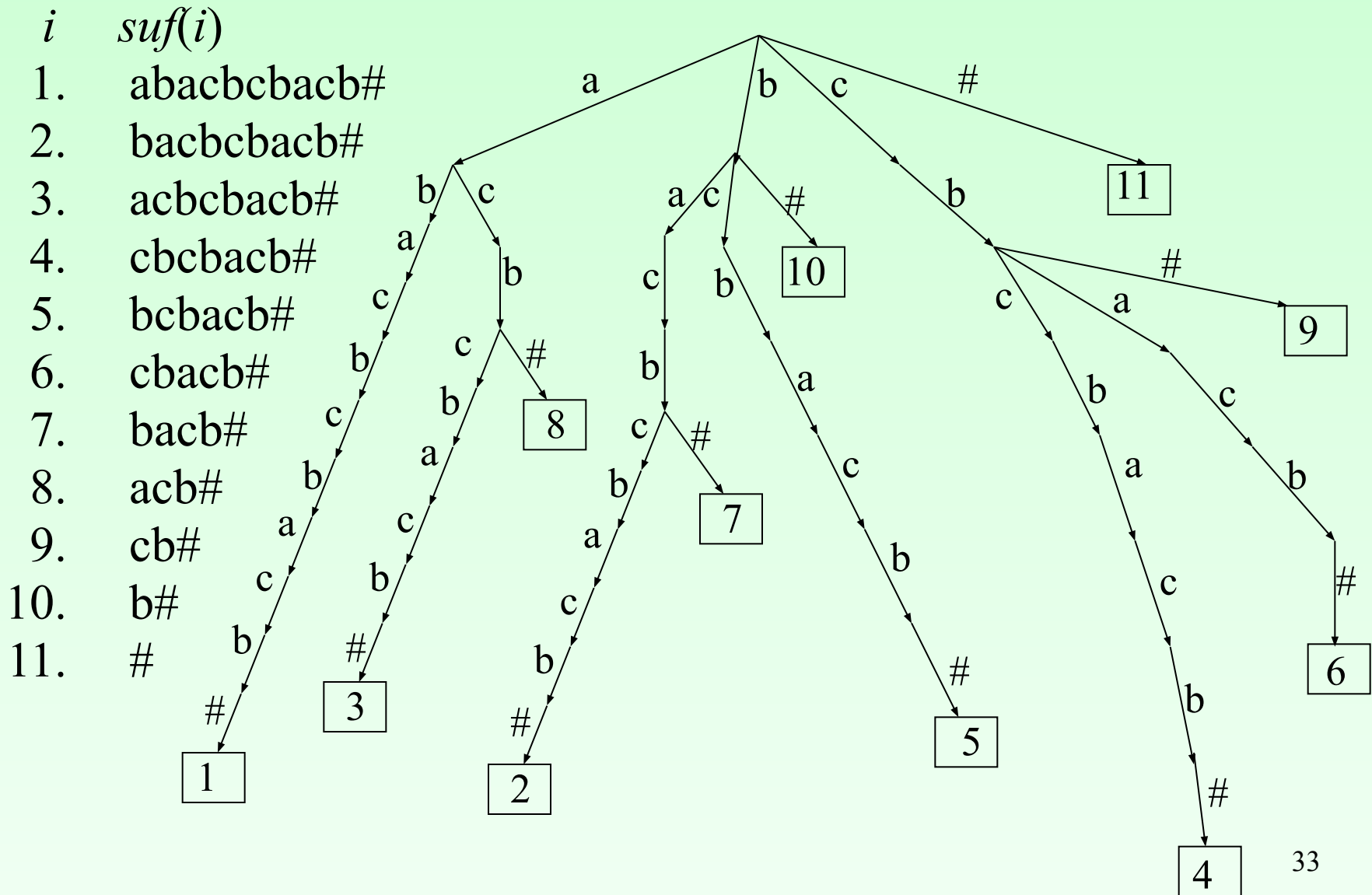


Пример компактного префиксного дерева для  $T\# = abacbcacbc\#$

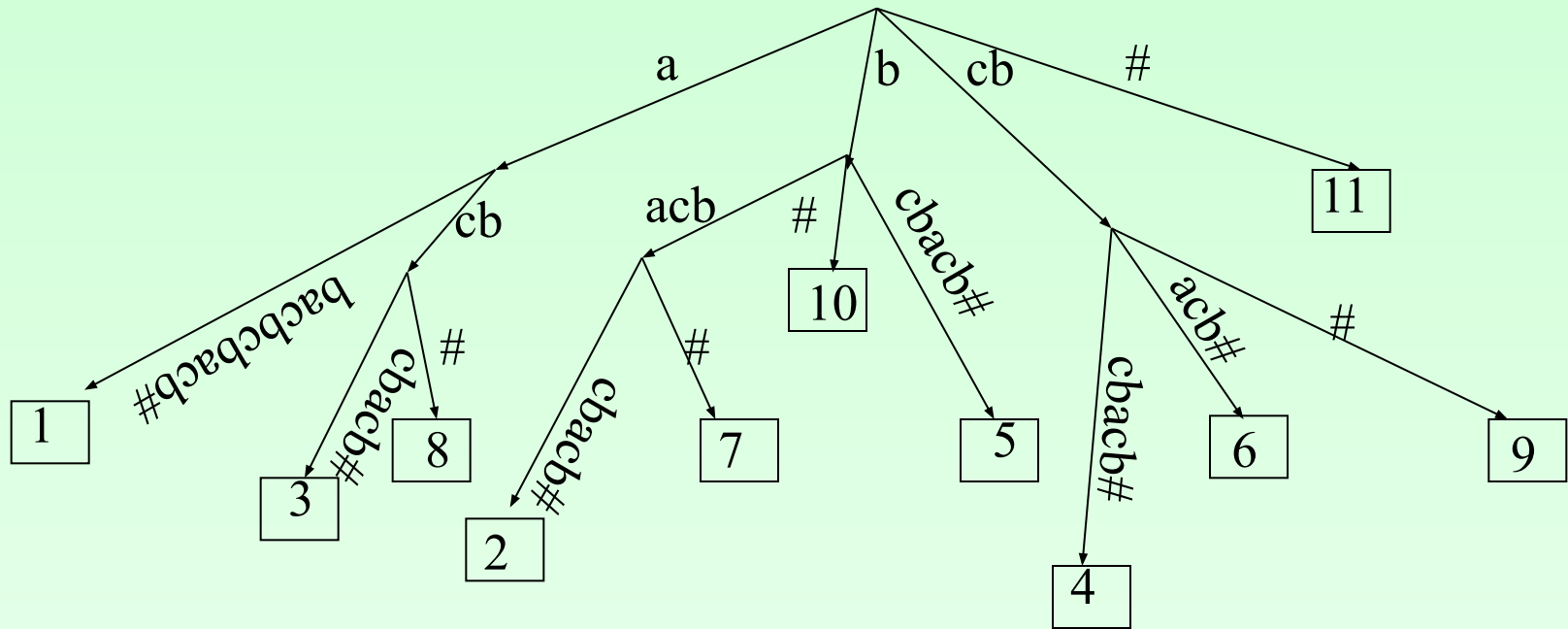




Пример дерева всех суффиксов для  $T\# = abacbcbacb\#$



# Суффиксное дерево для $T\# = abacbcabcb\#$



## Задачи, решаемые с помощью суффиксного дерева:

- Вычисление параметров полного частотного спектра;
- Поиск образца;
- Последовательный поиск множества образцов;
- Поиск образца во множестве строк;
- Наибольшая общая подстрока двух строк;
- Общие подстроки более чем двух строк;
- Задача загрязнения ДНК. Даны строки  $S_1$  и  $S_2$ :  $S_1$  – вновь расшифрованная ДНК,  $S_2$  – комбинация источников возможного загрязнения. Найти все подстроки  $S_2$ , которые встречаются в  $S_1$  и длина которых не меньше заданного  $l$ ;
- Суффиксно-префиксные совпадения всех пар строк (из заданного множества строк);
- Обнаружение всех «нерасширяемых» повторов;
- Задача о наибольшем общем «продолжении». Найти длину наибольшего общего префикса  $i$ -го суффикса строки  $S_1$  и  $j$ -го суффикса строки  $S_2$
- Выявление всех «нерасширяемых» палиндромов.