1. Естественные и формальные языковые системы.

Трактовка языка в широком смысле.

Единого и достаточно содержательного определения понятия "язык", по-видимому, не существует. Одна из причин – быстрый рост числа объектов, к которым применяется этот термин ("язык пчел", "язык дельфинов", "язык программирования", "язык запросов" (в информационных системах), "генетический язык" и др.).

С точки зрения лингвиста:

"**Язык** – это любая знаковая система". Знак – это условное обозначение некоторого предмета, свойства, отношения, явления.

Знаковая система – совокупность знаков, организованная определенным образом.

Семиотика – наука, изучающая свойства знаков и знаковых систем (как естественных, так и формальных).

Основные признаки, характеризующие представления экспертов о языке:

1) язык – средство коммуникации и выражения мыслей (функциональный аспект);

2) стихийность возникновения и развития (эволюционный аспект);

3) дискретность (внутренняя членимость сообщения);

4) иерархичность (комбинирование более крупных единиц из исходных мелких: морфемы – слова – предложения и т.д.);

5) открытость (возможность составления большого (потенциально бесконечного) числа осмысленных сообщений).

дополнительные (реже упоминаемые) признаки:

6) отсутствие жесткой однозначной связи между обозначением и обозначаемым (один и тот же объект может иметь несколько обозначений (явление синонимии) и одно и то же обозначение может быть использовано для разных объектов (явление омонимии));

7) экономичность кода (длина сообщения должна быть пропорциональна количеству информации в этом сообщении).

С точки зрения математика:

Пусть **Σ** – непустое конечное множество символов (алфавит);

строка ***W*** (цепочка символов, слово) – последовательность символов из Σ;

$|W|$ – длина цепочки W; e – пустая цепочка ($|e| = 0$);

Σ* – множество всех слов в алфавите Σ, включая e.

Язык *L* над алфавитом Σ – произвольное множество цепочек в Σ (т.е. $L \subseteq \Sigma^*$).

Конкатенацией языков L_1 и L_2 называется язык

$L_1 L_2 = \{ \alpha\beta : \alpha \in L_1, \ \beta \in L_2 \}$.

L* **: итерация языка *L*** :

$L^0 = \{ \varepsilon \}$,

$L^n = LL^{n-1}$ для $n \geq 1$,

$$L^* = \bigcup_{n \geq 0} L^n$$

Формальные языки

чаще всего задаются в виде некоторой схемы порождения (порождающей или формальной грамматики), которая позволяет получить все цепочки данного языка и только их. Обычно порождающая грамматика представляется в виде четверки

$G = (\Sigma, N, P, S)$, где

Σ – алфавит терминальных символов, из которых составляются "предложения" языка ($L(G) \subseteq \Sigma^*$);

N – алфавит нетерминальных символов (или переменных);

$\Sigma \cap N = \varnothing$;

P – конечное множество правили вывода вида $\alpha \to \beta$, где $\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*$, $\beta \in (N \cup \Sigma)^*$;

S – выделенный символ из N, называемый начальным (или исходным).

Формальные грамматики были введены Хомским (1956г). Им же была построена иерархия грамматик 4 типов.

Пример

Пусть $G = (\{a,b,c\}, \{A,B,S\}, P, S)$,

где правила вывода P имеют вид:

$S \to AB$, $A \to a$, $A \to ac$, $B \to b$, $B \to cb$.

Данная грамматика позволяет получить всего 4 вывода терминальных строк:

(1) $S \to AB \to aB \to ab$

(2) $S \to AB \to aB \to acb$

(3) $S \to AB \to acB \to acb$

(4) $S \to AB \to acB \to accb$

$L(G) = \{ab, acb, accb\}$.

Для строки acb имеются два разных вывода.

Конечные автоматы – средство распознавания «слов» языка

Детерминированный конечный автомат – это пятерка

$M = (S, \Sigma, \delta, s_0, F)$, где

S – конечное множество состояний;

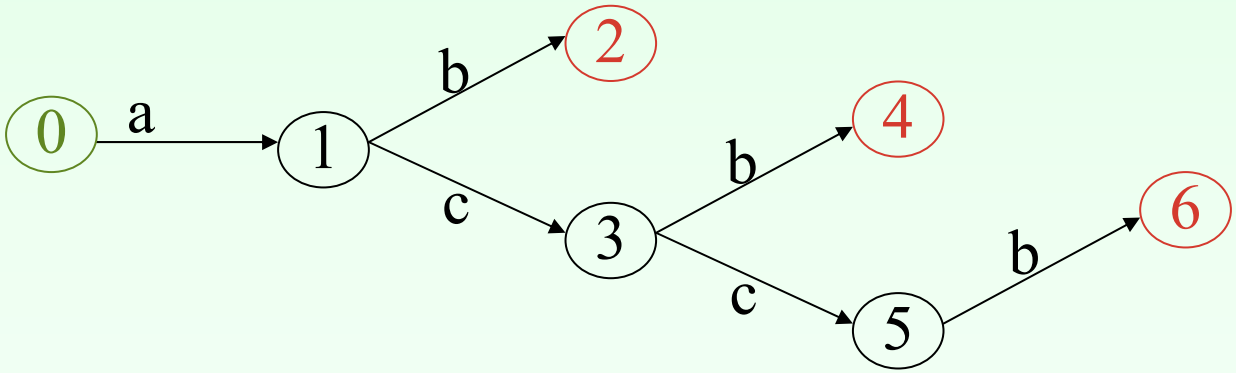
Σ – алфавит;

$\delta : S \times \Sigma \to S$ – функция переходов;

$s_0 \in S$ – выделенное начальное состояние;

$F \subseteq S$ – множество заключительных состояний;

Пример ДКА, допускающий {ab, acb, accb}.



Основные задачи

- **поиск образцов;**
- **восстановление структуры текста: выявление повторов (периодичностей, симметрий …) в тексте;**
- **сравнение последовательностей: разные определения расстояний и мер близости между последовательностями;**
- **сложность текста.**

2. Алгоритмы поиска точно заданных образцов

Дано: $P = p_1 p_2 \ldots p_m$ – образец, $T = t_1 t_2 \ldots t_N$ – текст,

$t_i \in \Sigma, p_j \in \Sigma, 1 \le i \le N, 1 \le j \le m, m < N.$

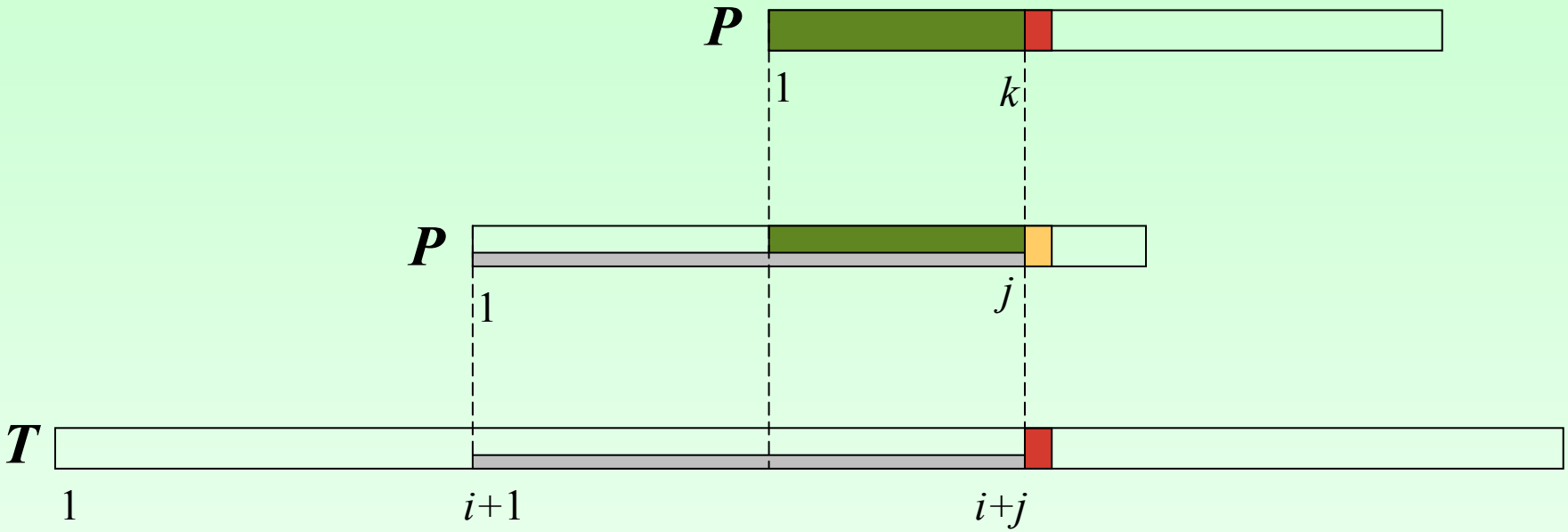
Задача: обнаружить все вхождения образца *P* в текст *T*.

Пример. Σ = {abcd}, *P* = aba, *T* = bbabacabcabad; *m* = 3, *N* = 13.

Образец входит в текст в 3 и 10 позиции. **Прямой алгоритм: 18 сравнений:**

T	=	b	b	a	b	a	c	a	b	c	a	b	a	d	
		a	b	a											1 сравнение
			a	b	a										1 сравнение
				a	b	a									3 сравнения, успех!
					a	b	a								1 сравнение
						a	b	a							2
							a	b	a						1
								a	b	a					3 сравнения!
									a	b	a				1
										a	b	a			1
											a	b	a		3, успех!
												a	b	a	1

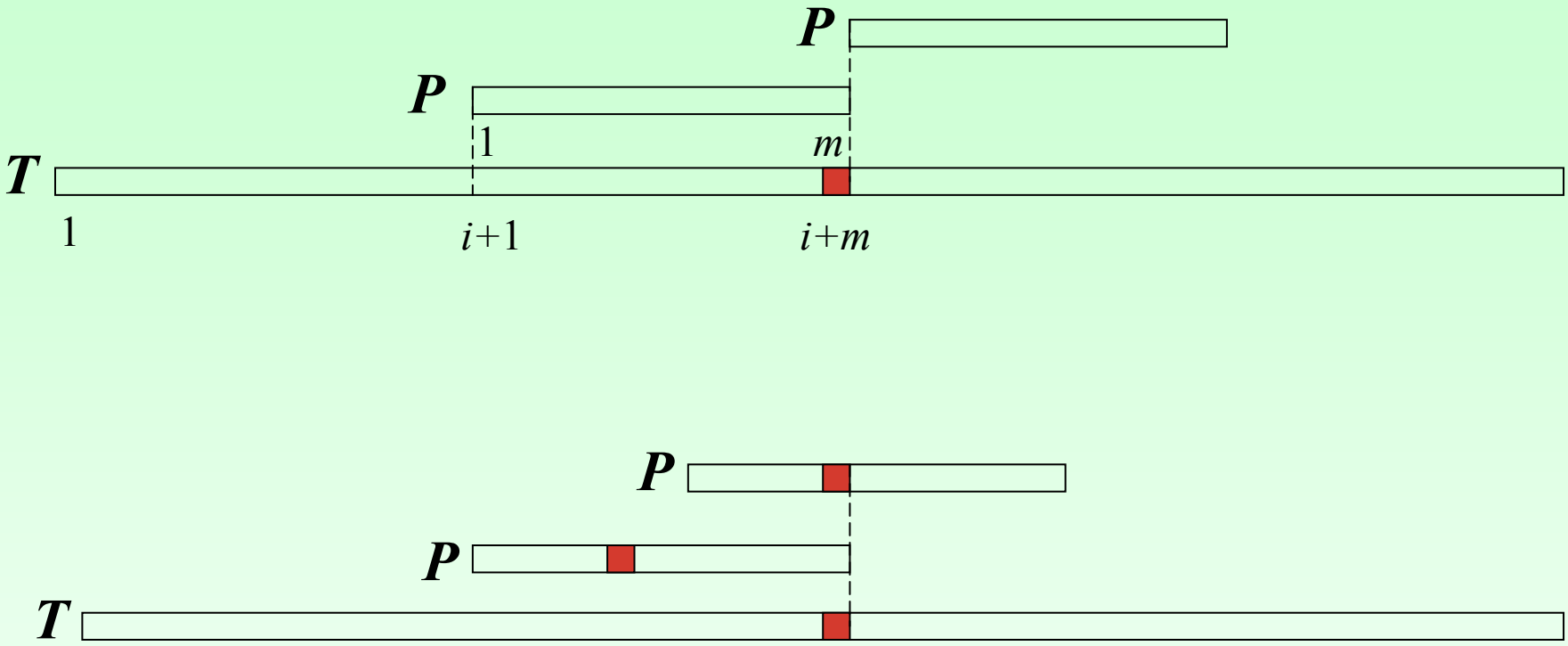
Алгоритм Кнута-Морриса-Пратта



Здесь $t_{i+1} \ldots t_{i+j} = p_1 \ldots p_j$, но $t_{i+j+1} \neq p_{j+1}$.

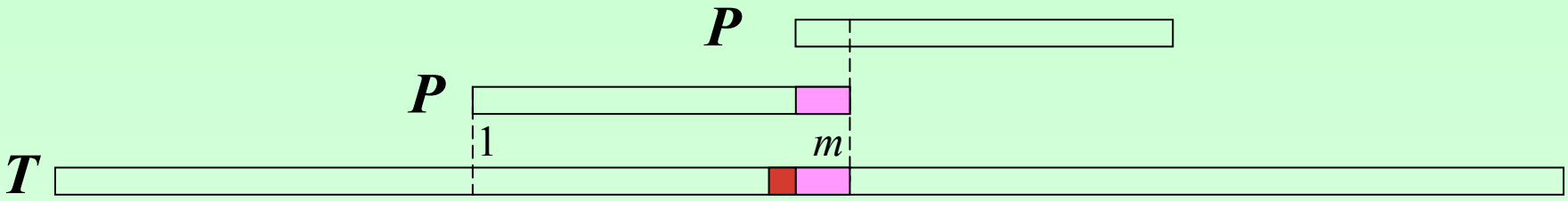
Если максимальный суффикс цепочки $p_1 \ldots p_j$, являющийся одновременно префиксом этой цепочки имеет длину k ($p_1 \ldots p_k = p_{j-k+1} \ldots p_j$, но $p_{k+1} \neq p_{j-k}$), можно переходить к сравнению символа t_{i+j+1} с p_{k+1}.

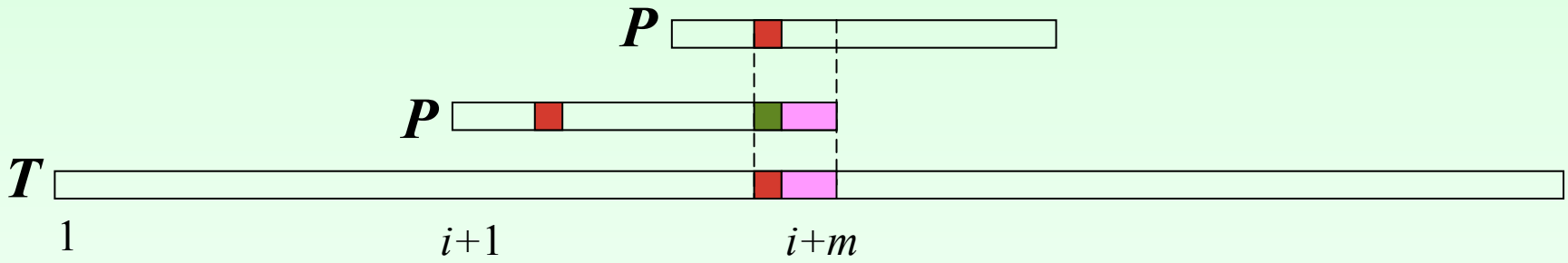
Алгоритм Бойера-Мура 1. Правило «плохого символа».



Правило «плохого символа».

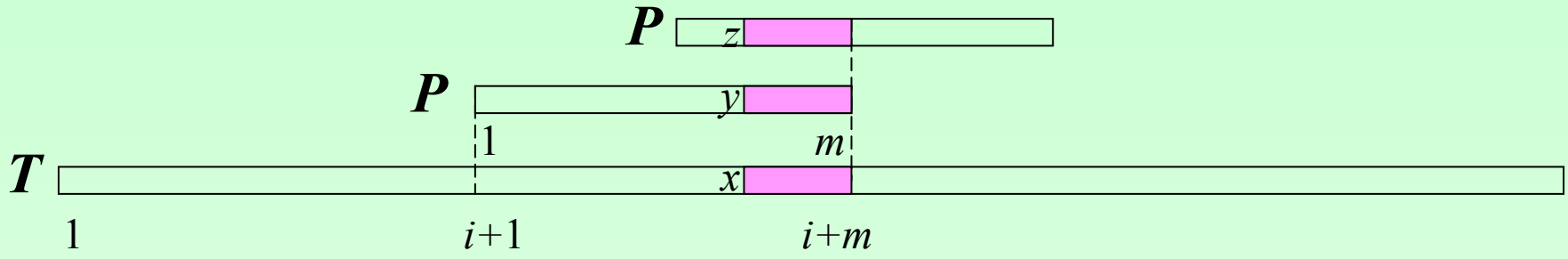
Алгоритм Бойера-Мура 1. Правило «плохого символа».





$$\delta_1(a) = \begin{cases} m - j, & \text{где} \quad j = \max\{i \mid p_i = a\}, \\ m, & \text{если} \quad a \notin \{p_1, \ldots p_m\}. \end{cases} \qquad a \in \Sigma$$

Алгоритм Бойера-Мура 2. Правило «хорошего суффикса».



$$\delta 2\ (j) = j + 1 - rpr(j), \text{ где } 1 \le j \le m$$

$$rpr(j) = \max\{k \mid (P[j+1:m] = P[k:k+m-j-1]) \quad and \quad ((k \le 1)\ or\ (p_{k-1} \ne p_j)\}$$

Алгоритм Shift-And

$$R[i,j] = \begin{cases} 1, & \text{если } P[1:i] = T[j-i+1:j], \\ 0, & \text{в остальных случаях.} \end{cases}$$

R[*m*,*j*] = 1 означает наличие образца *P* в (*j* – *m* + 1)-й позиции текста *T*.

$$R[i+1,j+1] = \begin{cases} 1, & \text{если } R[i,j] = 1 \text{ и } p_{i+1} = t_{j+1}, \\ 0, & \text{в остальных случаях}. \end{cases}$$

Пример. Пусть Σ = {a,b,c}, p=aabac, T=aabaacaabacab.

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

R

		a	a	b	a	a	c	a	a	b	a	c	a	b
	1	1	1	1	1	1	1	1	1	1	1	1	1	1
a	0	1	1	0	1	1	0	1	1	0	1	0	1	0
a	0	0	1	0	0	1	0	0	1	0	0	0	0	0
b	0	0	0	1	0	0	0	0	0	1	0	0	0	0
a	0	0	0	0	1	0	0	0	0	0	1	0	0	0
c	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0

Алгоритм Карпа-Рабина

Задаем порядок символов в Σ, т.е. взаимнооднозначное отображение $ns : \Sigma \to [0.. |\Sigma| - 1]$. Пусть $s = |\Sigma|$. Тогда

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

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

Рекуррентное хеширование:

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

Схема Горнера вычисления H:

$$H(P) = (\ldots(((ns(p_1) \times s + ns(p_2)) \times s + ns(p_3)) \times s + \ldots + ns(p_{m-1})) \times s + ns(p_m).$$

Пример. Σ = {acgt}, *P* = acat, *T* = ggacataccagac;

$H(P) = 1 \times 4^3 + 2 \times 4^2 + 1 \times 4^1 + 4 = 104;$

$H(T[1:4]) = 3 \times 4^3 + 3 \times 4^2 + 1 \times 4^1 + 2 = 246;$

$H(T[2:5]) = 3 \times 4^3 + 1 \times 4^2 + 2 \times 4^1 + 1 = 217 = (246 - 3 \times 4^3) \times 4 + 1;$

$H(T[3:5]) = 1 \times 4^3 + 2 \times 4^2 + 1 \times 4^1 + 4 = 104 = (217 - 3 \times 4^3) \times 4 + 4;$

Алгоритм Ахо-Корасик.

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

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

Текст $\boldsymbol{T} = t_1\ t_2\ \ldots\ t_N$, $t_k \in \Sigma$, $1 \le k \le N$.

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.

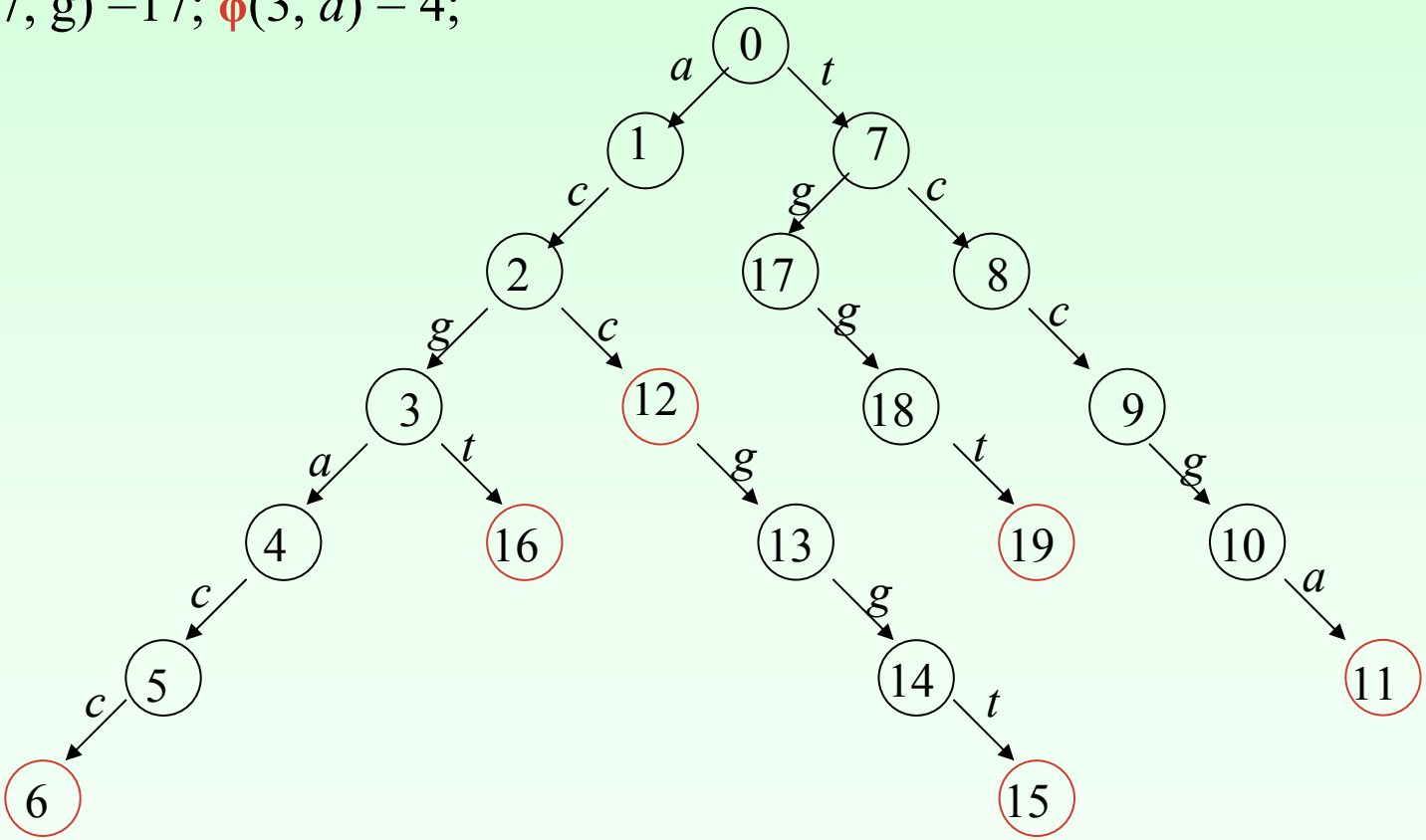
   Сначала строится "машина идентификации цепочек" ***Mp***. Работа машины ***Mp*** описывается тремя функциями: функцией переходов $\varphi(s,a)$ (s – состояние машины, $a \in \Sigma$), функцией отказов $f(s)$ и функцией выходов $o(s)$.

Алгоритм Ахо-Корасик.

Функция переходов φ(s,*a*)=s', если существует выходящее из s ребро, помеченное символом "*a*" и связывающее состояния s и s'; в противном случае φ(s,a) = "fail" (ситуация, обозначаемая термином "отказ").

Пример. Пусть Σ = {a,c,g,t}; P = {acgacc, tccga, accggt, acgt, acc, tggt};

φ(7, g) =17; φ(3, *a*) = 4;



Алгоритм Ахо-Корасик.

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

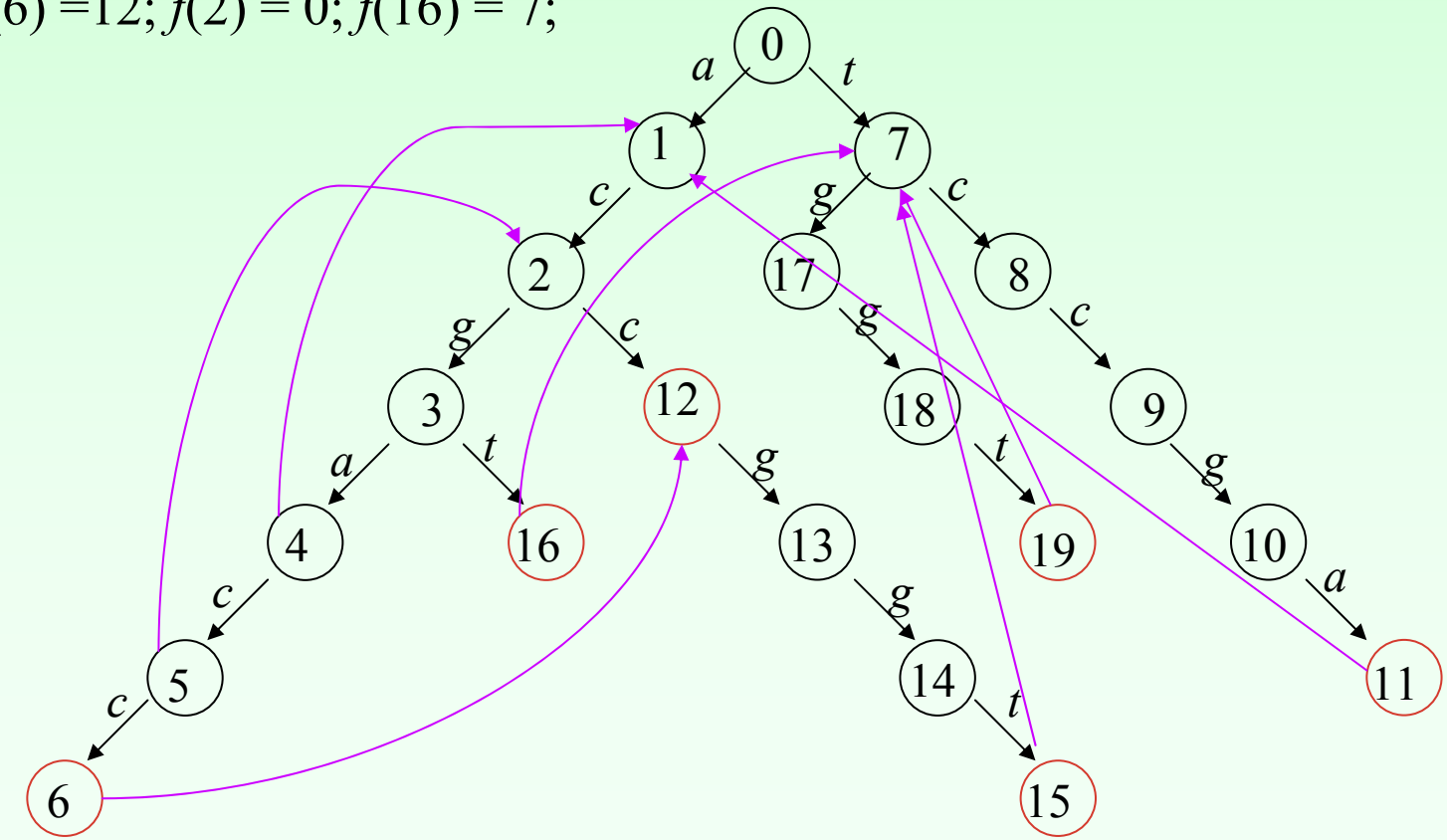
Metka : Если φ(s",a)<> fail, то f(s)= φ(s",a); o(s) := o(s) ∪ o(f(s)) ,

иначе s" := f(s"); goto Metka.

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

Пример. Пусть Σ = {a,c,g,t}; P = {acgatc, tccga, accggt, acgt, acc, tggt}; o(6)={1,4};

$f(6) = 12; f(2) = 0; f(16) = 7;$



3. Классификация повторов.

- **По способу их прочтения и использованию переименований:**

Повторы – элементарные структурообразующие компоненты текстов. Будем называть повтором (в широком смысле) пару фрагментов текста, совпадающих с точностью до переименования элементов алфавита и (возможно), изменения направления считывания элементов в одном из фрагментов на противоположное.

В соответствии с этим будем различать следующие типы повторов:

– прямые : … AGTTC … AGTTC…

– симметричные: … $\overleftarrow{\text{AGTTC}}$ … $\overrightarrow{\text{CTTGA}}$…

– с точностью до подстановки на элементах алфавита: секвентные переносы в музыкальных произведениях; замены $0 \to 1$ и $1 \to 0$ в двоичных словах; замены $A \to T$, $T \to A$, $C \to G$, $G \to C$, связанные с отношением комплементарности в ДНК-последовательностях:

– прямые комплементарные: … AGTTC… TCAAG…

– симметричные комплементарные: … $\overrightarrow{\text{AGTTC}}$… $\overleftarrow{\text{GAACT}}$…

• **По наличию искажений:**

Повторы могут быть

совершенные:

… AGTTC … AGTTC…

и несовершенные (с заменами, вставками, делециями):

… AGTTC … AATTC … (замена),

… AGTTC … AGTTTC… (вставка),

в том числе с точностью до агрегирования:

… AGTTC … GATCT …

(совпадают при заменах {A,G} → Pu, {C,T} → Py).

- **По характеру расположения в тексте:**

будем различать повторы

разнесенные

… AGTTC … AGTTC…

тандемные … AGTTC AGTTC…

с наложением : … AGTTCAGTTCAGTTC …

Примеры повторов в генетических текстах

1) Участки с аномальным нуклеотидным составом:

   (A,T)-богатые, (C,G)-богатые и т.п.

   Тип повторов – как правило, несовершенные.

2) Участки микросателлитной ДНК: периодичности с малой длиной периода (обычно 2,3) и достаточно высокой кратностью повторений ($(TA)^n$, $(CAG)^n$… n – порядка 10 и выше). Тип повторов: обычно совершенные.

3) Тандемные повторы со средней и большой длиной периода (L ~10 и выше) и кратностью повторений, уменьшающейся "в среднем" с увеличением длины периода. Тип повторов: совершенные и несовершенные.

4) Разнесенные повторы значительной длины ($L \sim 10^2$), совершенные (например, длинные концевые повторы в геномах некоторых микроорганизмов) и несовершенные (повторы крупных блоков в кодирующих участках некоторых генов).

5) Повторы регуляторных и функциональных единиц (как правило, несовершенные):

– рибосомные сайты связывания, промоторы и терминаторы транскрипции у прокариотов ($L \sim 10 – 10^2$);

– сайты связывания транскрипционных факторов у эукариотов ($L \sim 10 – 20$);

– гены и псевдогены ($L \sim 10^3 – 10^4$);

– повторы отдельных генов (рибосомальные, гистоновые…), $L \sim 10^3 – 10^4$;

– мобильные элементы ($L \sim 10^3 – 10^4$);

6) Блоки из регуляторных и функциональных единиц с вкраплением некодирующих участков, часто соответствующих отдельным транскриптам ($L \sim 10^4$);

Следует отметить, что многие регуляторные и функциональные единицы, в свою очередь, построены по принципу тандемной повторности. Приведем для иллюстрации выравнивание, соответствующее тандемным повторам ("итеронам"), расположенным в зоне начала репликации бактериофага λ:

IT-1:	39034:	ATCCCTCAAAACGAGGGAA	A ↵
IT-2:	39054:	ATCCCCTAAAACGAGGGAT	AAAAC ↵
IT-3:	39078:	ATCCCTCAAATTGGGGGAT	TGCT ↵
IT-4:	39101:	ATCCCTCAAAACAGGGGGA	

Здесь значок ↵ означает, что нижеследующая строка непосредственно продолжает предыдущую. Интересно отметить своего рода эффект "локальной фрактальности": палиндромно-шпилечные структуры внутри итеронов приводят к возникновению аналогичных (но более сильных) структур между итеронами:

AATCCCCTAAAACGAGGGATAAAACATCCCTCAAATTGGGGGATT

IT-2 IT-3

4. Представление текста в терминах повторов.

Полный частотный спектр текста. Пусть

Σ – конечный алфавит;

S – текст, составленный из элементов Σ;

$N = |S|$ – длина текста;

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

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

Назовем l-граммой связную цепочку текста, содержащую l символов.

Всякий текст $S = a_1\, a_2\, a_3\, a_4\, a_5 \ldots a_N$

можно представить в виде последовательности l-грамм ($l = 1,2,\ldots,N$), выделяемых скользящим окном ширины l , движущимся вдоль текста с шагом в 1 символ.

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

С учетом возможных повторений число различных l-грамм $M_l \le N - l + 1$.

Назовем *частотной характеристикой l-го порядка* текста S совокупность элементов $\Phi_l(S) = \{\phi_{l1}, \phi_{l2}, \ldots, \phi_{l M_l}\}$

где $\phi_{l M_i}$ $(1 \le i \le M_l)$ есть пара :

<*i*-я *l*-грамма – x_i, частота ее встречаемости в тексте – $F_l(x_i)$ >.

Пусть $l_{max}(S)$ – наибольшее значение l, при котором в тексте S еще содержатся повторяющиеся l-граммы.

Совокупность частотных характеристик

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

назовем *полным частотным спектром текста S*.

По нему текст S может быть восстановлен однозначно.

На практике чаще используют усеченный спектр

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

куда не входят $\Phi_{l\,max+1}(S)$ и $\Phi_{l\,max+2}(S)$, а $\Phi^*_l(S)$ отличаются от $\Phi_l(S)$ лишь отсутствием l-грамм с единичной частотой встречаемости.

Пример. Пусть S = caabcabbca.

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

$\Phi^*_1(S) = \{<a, F(a) = 4>; <b, F(b)= 3>; <c, F(c) = 3>\};$

$\Phi^*_2(S) = \{<ca, F(ca) = 3>; <ab, F(ab)= 2>; <bc, F(bc) = 2>\ldots\};$

$\Phi^*_3(S) = \{<bca, F(bca) = 2>\ldots\};$

Позиционную информацию целесообразно выдавать лишь для повторов значительной длины.

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 $(1 \le r \le n = |\Sigma|)$ можно пользоваться следующей оценкой: .

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

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

M_l — размер словаря l-грамм.

Он фигурирует в определении комбинаторной сложности текста.

$F_l^{\max} = \max\limits_{1 \le i \le M_l} F_l(x_i)$ — максимальное значение частот встречаемости l-грамм в тексте $(1 \le l \le l_{\max})$;

$F_l^{\min} = \min\limits_{1 \le i \le M_l} F_l(x_i)$ — минимальное значение частот встречаемости l –грамм в тексте $(1 \le l \le l_{\max})$; представляет интерес лишь при малых значениях l; при больших, обычно $F_l^{\min} = 1$.

E_l^k — Число различных l-грамм, каждая из которых встречается в тексте ровно k раз ($k = 1,2,\ldots,N - l + 1$);
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 $N / n^l >> 1$, можно трактовать как аномальный эффект.

Имеют место простые соотношения, связывающие основные параметры:

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

С помощью частотного спектра текста можно:

а) сравнить реальный текст с его "случайной копией" с тем же составом элементов ($\Phi_1(T)$), но с разрушенными связями между символами; выявить при этом аномально длинные повторы, аномально частые и аномально редкие цепочки, цепочки, сохраняющие частоту при расширении

(см. переход от ***q*** к ***qu*** в английском) и т.п.;

б) получить оценки переходных вероятностей при построении марковских моделей текста

(необходимо, чтобы размер текста был достаточно велик);

в) вычислить энтропийные характеристики *l*-го порядка и оценки избыточности текста (Шеннон);

г) получить численные оценки близости двух (или большего числа) текстов в ситуации, когда перестают работать методы выравнивания.

5. Алгоритмы отыскания совершенных повторов.

5.1. Метод лексикографической сортировки

Лексикографический порядок для двух слов $u = u_1 \ldots u_p$ и $v = v_1 \ldots v_q$ определяется следующим образом:

$u \le v$, если $u = v$ или $(\exists\, j$, такое что $u_j < v_j$ и $u_i = v_i$ для всех $i < j)$ или $(p < q$ и для всех $i \le p \;\; u_i = v_i)$.

Предполагается, что порядок символов алфавита задан.

В результате процедуры сортировки одинаковые l-граммы оказываются собранными в группы "соседних" l-грамм. Остается подсчитать количество l-грамм в таких группах. Трудоемкость метода — $O(l \cdot N)$ (в предположении, что $|\Sigma| << N$. Затраты памяти — $O(l \cdot N)$).

5.2. Метод, основанный на хешировании

символьных цепочек позволяет вычислить $\Phi_l(T)$ с временными затратами $O(l \cdot N)$ в среднем и затратами памяти $O(N \cdot log\ N)$). Трудоемкость «рекуррентного хеширования» — $O(N)$.

Хеширование – это отображение, которое ставит в соответствие произвольной l-грамме текста x_i $(1 \le i \le N - l + 1)$ адрес оперативной памяти $h(x_i)$, в котором хранится информация об x_i.

Примером простейшего отображения является представление l-граммы $x_i = a_{i1}a_{i2}\ldots a_{il}$, где $a_{im} \in \Sigma$ $(m = 1 \div l)$ в q-ичной системе счисления, где $q = |\Sigma|$. Элементам алфавита $\{a_0, a_1, \ldots, a_{q-1}\}$ при этом ставятся в соответствие числа $\{0,1,\ldots q-1\}$, а цепочке x_i длины l соответствует число

$$h_1(x_i) = k_1 q^{l-1} + k_2 q^{l-2} + \ldots + k_l q^0 = \sum_{i=1}^{l} k_i q^{l-i},$$

где k_m – числовой эквивалент символа a_{im} из цепочки x_i $(0 \le k_i \le q-1)$.

Недостаток этого отображения – слишком большой (порядка $|\Sigma|^l$) диапазон изменения чисел $h_1(x_i)$, что приводит к нерациональному расходу памяти (сильно разреженный массив адресов).

Достоинство – отображение h_1 является взаимно-однозначным и достаточно просто вычислимым.

Компромисс по памяти и по времени может быть достигнут, если

а) сузить диапазон изменения возможных значений $h(x)$ до величины, соответствующей реальному разнообразию сравниваемых объектов

(в случае l-грамм эта величина не превышает $N - l + 1$). Этому требованию удается удовлетворить, если

б) отказаться на начальном этапе от требования однозначности нумерующей функции (или функции расстановки) $h(x_i)$, вследствие чего могут возникнуть наложения

(ситуации, когда $h(x_i) = h(x_j)$ при $x_i \neq x_j$);

в) разделить в последующем наложившиеся объекты с помощью специальной техники (открытая адресация, перехеширование, использование списковых структур и т.д.).

Пример функции расстановки, допускающей наложения:

$$h_2(x_i) = K(x_i) \bmod M = K(x_i) - M \lfloor K(x_i)/M \rfloor$$

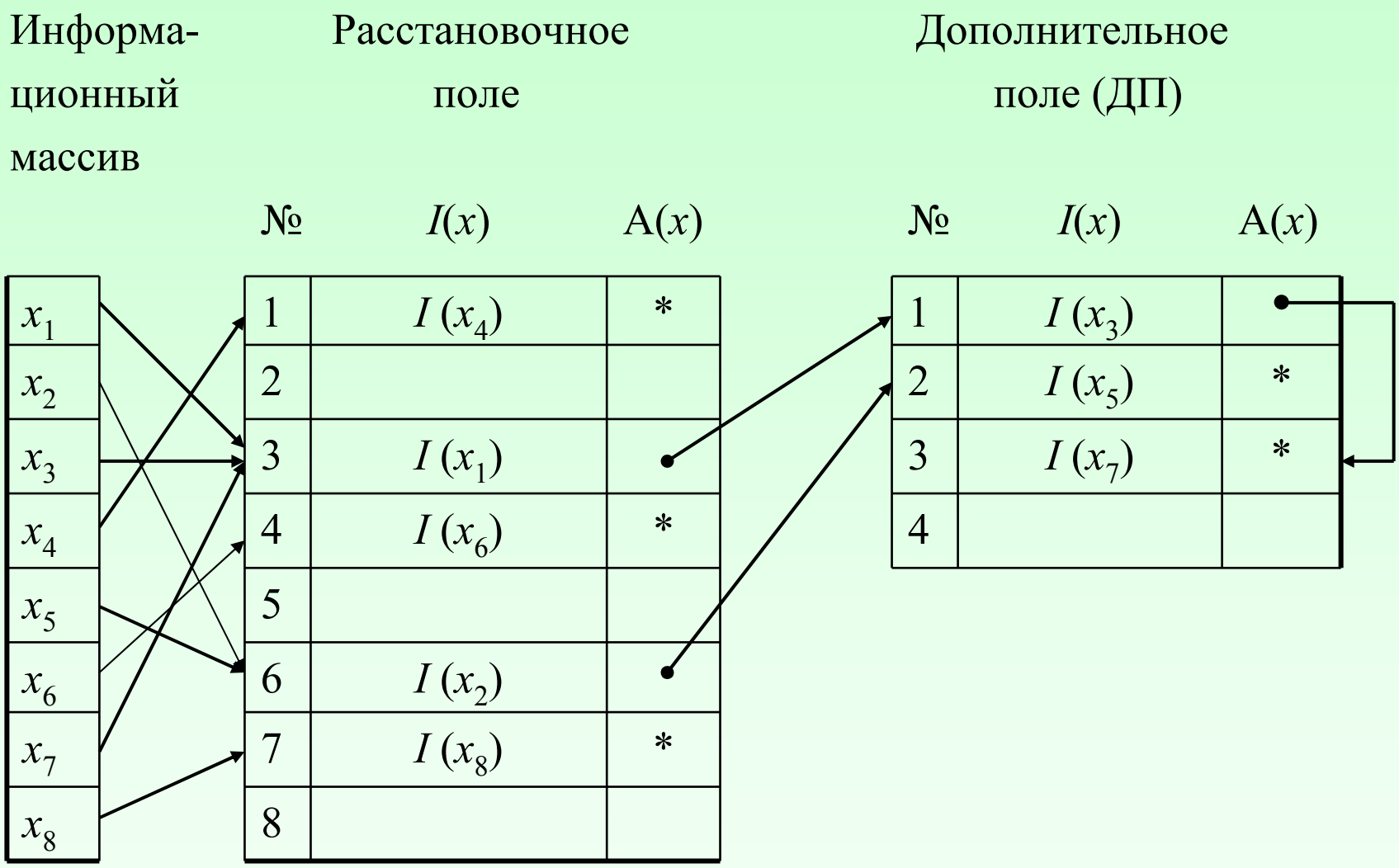
где $K(x_i)$ числовой код l-граммы x_i ,

M — оценка сверху разнообразия l-грамм в тексте T ($M_l < N - l + 1$);

$\lfloor Z \rfloor$ означает целое, ближайшее снизу к Z.

M желательно выбирать в виде простого числа.

Пример списковой схемы устранения наложений



Здесь $x = (x_1, x_2, \ldots x_8)$ — информационный массив, элементы которого хешируются в порядке их следования. Каждый элемент расстановочного поля делится на две части: информационную $I(x)$ и адресную A(x). В $I(x)$ заносится информация о самом первом объекте, распределенном по данному адресу (например, имя объекта и счетчик числа его вхождений в информационный массив). При наличии повторного обращения по данному адресу увеличивается на 1 значение счетчика, если объекты (старый и новый) совпали; в противном случае в A(x) указывается адрес первой свободной позиции в дополнительном поле, куда помещается наложившийся объект. Так поступают со всеми наложившимися объектами в основном поле. При наличии трех – (и более высокой кратности) наложений в расстановочном поле (см. объекты x_1, x_3, x_7) мы второй и все последующие объекты размещаем в дополнительном поле, связывая их отсылками (см. стрелки в ДП). Звездочка, стоящая в A(x) означает конец списка.

5.3. Префиксное и суффиксное деревья

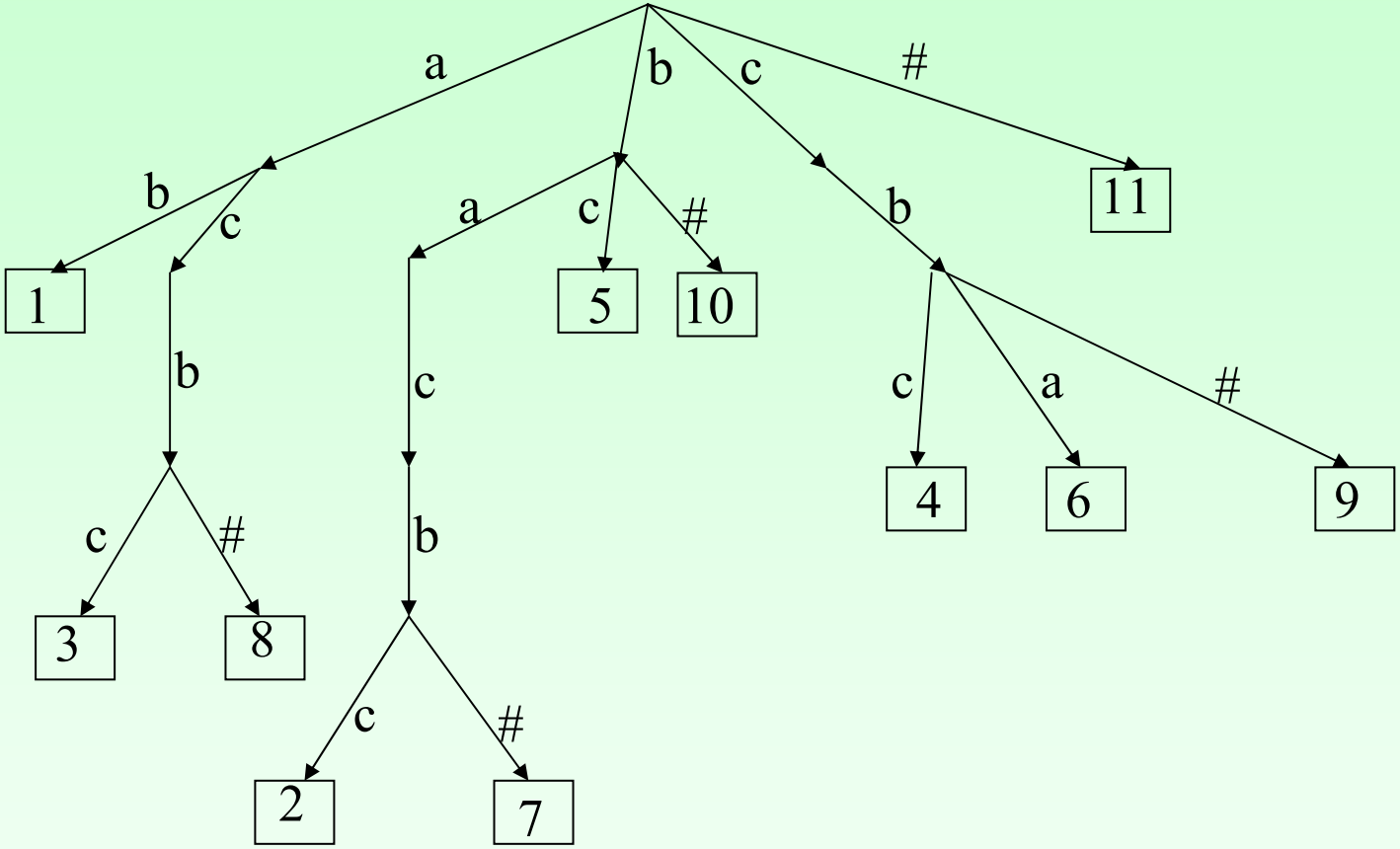
Если $v = xyz$, то x называют префиксом v, z – суффиксом, а y – подсловом. Оптимальные алгоритмы отыскания совершенных повторов основаны на построении префиксного дерева, суффиксного дерева или графа подслов текста (DAWG).

Первая конструкция принадлежит Вайнеру (Weiner P., 1973), вторая – Мак-Крейгу (McCreight, 1976), третья – коллективу авторов (A.Blumer, J.Blumer, A.Ehrenfeucht, et al., 1984). Все конструкции являются ***функционально эквивалентными*** 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. Чтобы $pr(i)$ был определен для всех позиций текста, T дополняют конечным маркером $t_{N+1} = \#$, ($\# \notin \Sigma$). Множество всех префикс-идентификаторов можно представить в виде ***префиксного дерева***. Ребра его помечены символами из $(\Sigma \cup \#)$, а листья – числами $1,2,\ldots,N+1$, однозначно соответствующими позициям в $T\#$.

Пример дерева префикс-идентификаторов для T# = abacbcbacb#

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

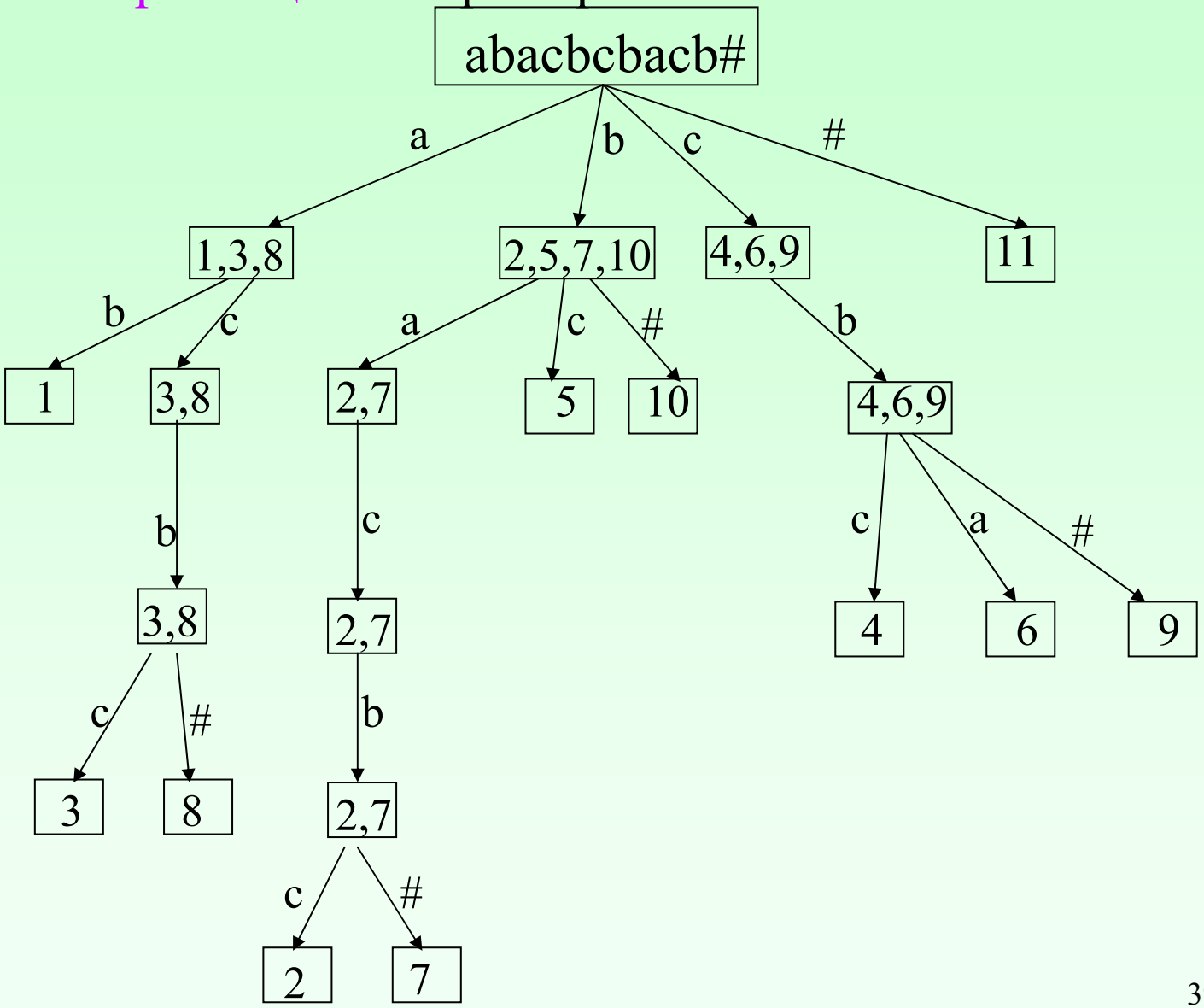


Алгоритм Мартинеца позволяет вычислить полный частотный спектр повторов с временными затратами $O(N \log N)$ в среднем и $O(N^2)$ в наихудшем случае.

На первом шаге проходим по тексту и сортируем его элементы по типам букв, связывая с каждым типом узел на первом уровне дерева, содержащий список позиций вхождений данного символа в текст. Если список состоит из одной позиции, то данная буква (а в общем случае цепочка символов, которая помечает путь из корня дерева в данный узел) однозначно идентифицирует указанную позицию текста (нет другой позиции с той же цепочкой). Соответствующий узел дальше не ветвится.

Если список состоит более чем из одной позиции (есть повторы), сортируем его по типам букв, следующих в тексте непосредственно за буквой (в общем случае – цепочкой), породившей данный список (сортировка биграмм). Образуем для каждой биграммы узел на втором уровне дерева и т.д. Процесс заканчивается, когда все списки становятся одноэлементными. Алгоритм очень прост по своей логике, но требует значительных затрат памяти для хранения позиционной информации и больших временных затрат в наихудшем случае.

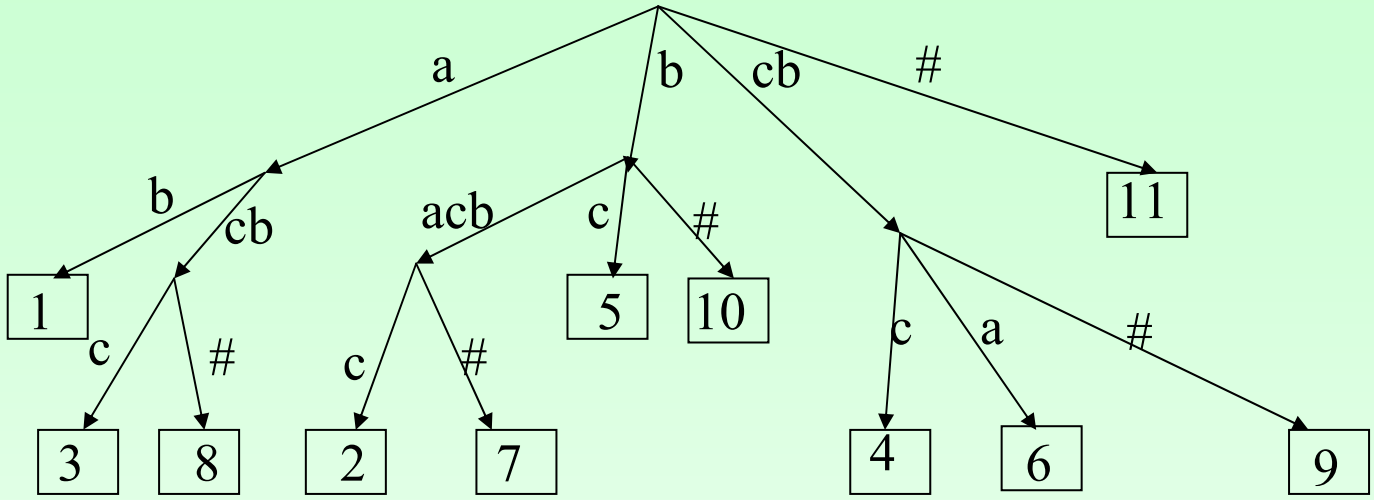
Алгоритм Мартинеца на примере T# = abacbcbacb#



Нетрудно видеть, что **конструкция Мартинеца** полностью соответствует **префиксному дереву**, но в последнем отсутствует позиционная информация во внутренних узлах. Отличаются и схемы построения: Мартинец строит дерево "по уровням" (первый, второй и т.д.); Вайнер – по цепочкам (префикс-идентификаторам), просматривая текст справа – налево (режим off-line!) и добавляя на каждом шаге новую цепочку (t_8, t_7, t_6 и т.д.).

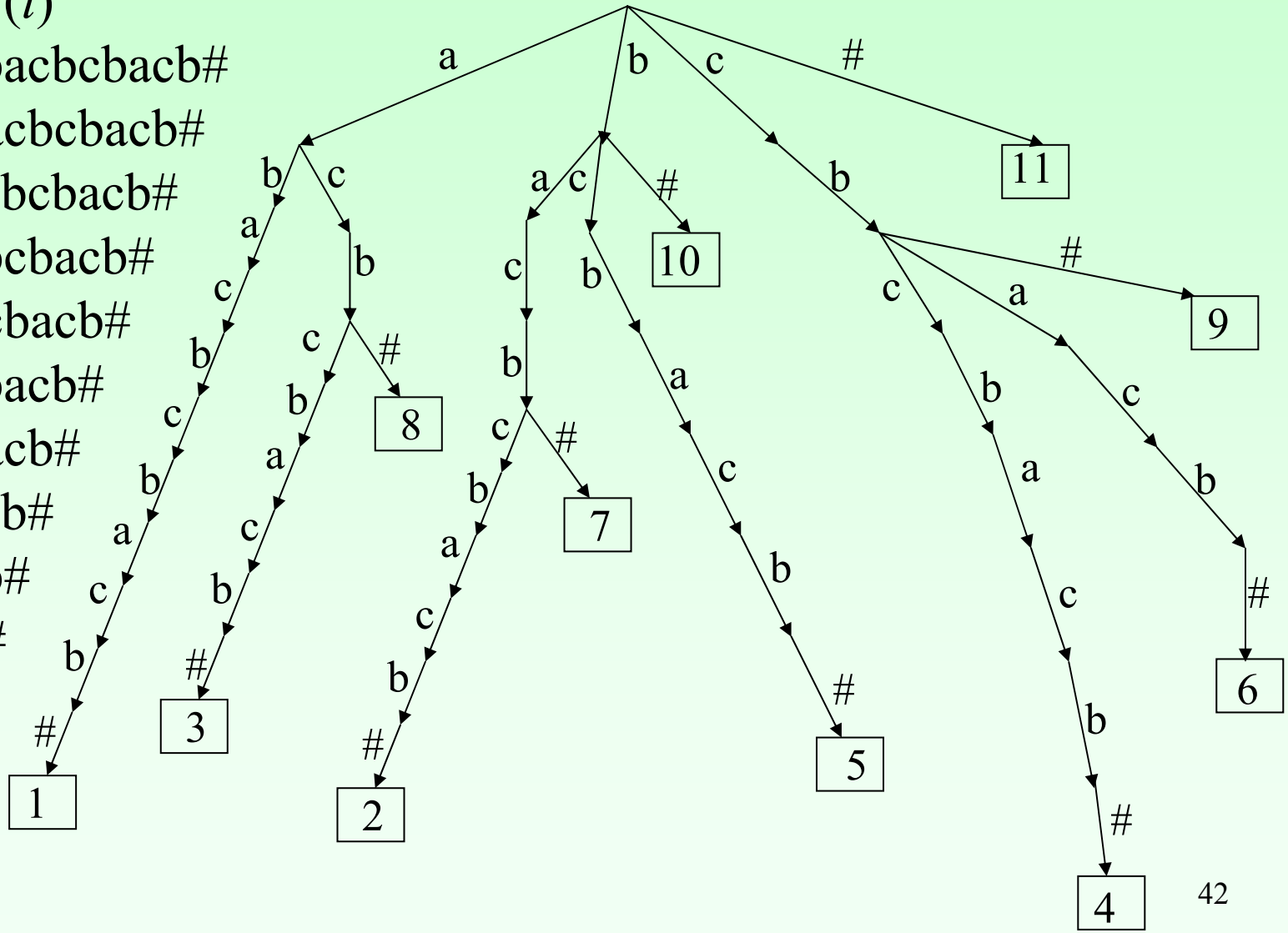
В общем случае префиксное дерево может иметь $O(N^2)$ узлов. К примеру, префиксное дерево для текста $T = a^k b^k a^k b^k z$, где a^k и b^k – k-кратные повторения символов a и b соответственно, содержит $k^2 + 6k + 2$ узлов (показать). Поэтому временные затраты на его построение в наихудшем случае также будут иметь порядок N^2. Однако число узлов может быть уменьшено путем ***компактизации*** *дерева*, сводящейся к замене всех цепных (неразветвляющихся) участков одним ребром, помеченным теперь цепочкой символов (в приведенном выше примере ребра ***a*** и ***b*** вместе с разделяющим их узлом будут заменены одним ребром, помеченным цепочкой ***ab***). Вайнер показал, что компактное префиксное дерево содержит не более $3N - 2$ узлов и может быть построено с *линейными временными затратами*. Однако логика алгоритма Вайнера достаточно сложна, что ведет к существенному росту мультипликативных констант в оценках трудоемкости и памяти. Это означает, что более простые алгоритмы (типа Мартинеца), хотя и содержат нелинейность в оценке трудоемкости, могут тем не менее конкурировать с оптимальными алгоритмами в достаточно значительном диапазоне длин текстов.

Пример компактного префиксного дерева для T# = abacbcbacb#

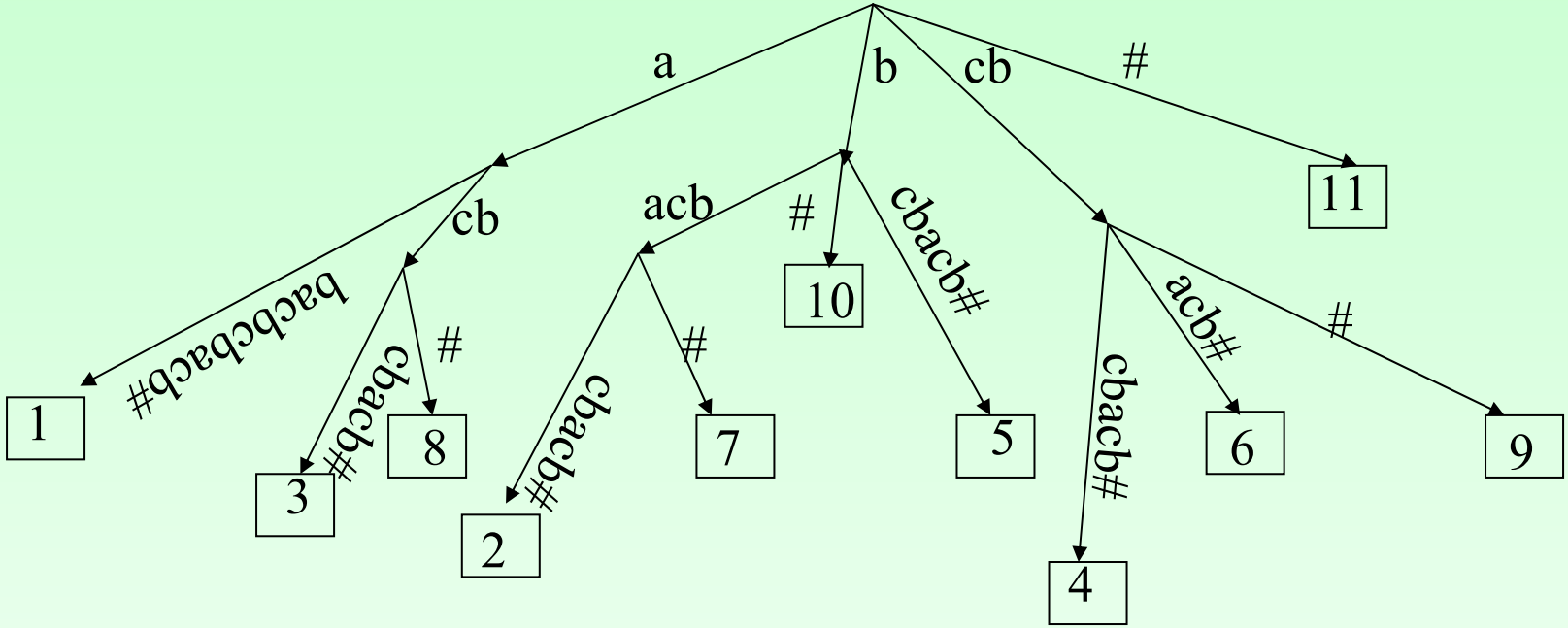


Пример дерева всех суффиксов для T# = abacbcbacb#

i	*suf*(*i*)
1.	abacbcbacb#
2.	bacbcbacb#
3.	acbcbacb#
4.	cbcbacb#
5.	bcbacb#
6.	cbacb#
7.	bacb#
8.	acb#
9.	cb#
10.	b#
11.	#



Суффиксное дерево для T# = abacbcbacb#



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.

6. Расстояния между строками и меры близости

Расстояние Хэмминга: число несовпадений символов.

Пример: u = abbaeac; $d_{хэм} = 2$.

v = abdaecc;

Редакционное расстояние. В простейшем виде расстояние $d(u,v)$ между строками u и v определяется как минимальное число операций типа "вставка", "устранение" или "замена" символа, переводящих одну строку в другую.

Пример: u = abbaeac; v = bdedac

$u \to v$:

a	b	b	a	e	a	c
	|				|	|
–	b	d	e	d	a	c

$d(u,v) = 4$ (1 дел., 3 зам).

$u \to v$:

a	b	b	a	e	–	a	c
	|			|		|	|
–	b	d	–	e	d	a	c

$d(u,v) = 4$ (2 дел., 1 вст., 1 зам).

Хроника введения расстояний и мер сходства, основанных на идеях динамического программирования.

Год	Наименование (меры, расстояния, процедуры сравнения)	Авторы	Предметная область
1965	Метрика Левенштейна	Левенштейн В.И.	теория связи (двоичные коды)
1968	процедуры нелинейной нормализации речи по темпу	Слуцкер Г.С.	распознавание речи
1968	ДП - метод	Винцюк Т.К.	распознавание речи
1969	дискретный вариант меры Слуцкера	Загоруйко Н.Г., Величко В.М.	распознавание речи
1970	мера сходства АМ-последовательностей	Needleman S.B., Wunch S.B.	молекулярная биология
1974	редакционное расстояние	Wagner R.A., Fisher M.J.	лингвистика
1974	эволюционное расстояние	Sellers P.	молекулярная биология

Схема динамического программирования для вычисления редакционного расстояния:

$d(0,0) = 0;$

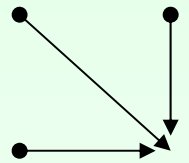
$d(i,0) = i,\ 1 \le i \le m;\ m = |u|;$

$d(0,j) = j,\ 1 \le j \le n;\ \ n = |v|;$

$$d(i,j) = \min \{d(i-1,\ j-1) + \gamma(u_i, v_j),\ d(i-1,\ j) + 1,\ d(i,\ j-1) + 1\}.$$

$$\gamma(u_i, v_j) = \begin{cases} 0, & u_i = v_j \\ 1, & u_i \neq v_j \end{cases}$$

	ε	a	b	b	a	e	a	c
ε	0	1	2	3	4	5	6	7
b	1	1	1	2	3	4	5	6
d	2	2	2	2	3	4	5	6
e	3	3	3	3	3	3	4	5
d	4	4	4	4	4	4	4	5
a	5	4	5	5	4	5	4	5
c	6	5	5	6	5	5	5	**4**



Меры сходства

а) Пусть T_1 и T_2 два текста.

Назовем совместной частотной характеристикой l-го порядка текстов T_1 и T_2 совокупность элементов

$$\Phi_l(T_1, T_2) = \{\phi_{l1}(T_1, T_2), \phi_{l2}(T_1, T_2), \ldots, \phi_{l M_l}(T_1, T_2)\}$$

где $M_l = M_l(T_1, T_2)$ — число l-грамм, общих для обоих текстов,

а элемент $\phi_{l\,i}$ $(1 \le i \le M_l)$ есть тройка:

<< i-я общая l-грамма – x_i,

частота ее встречаемости в T_1 – $F(T_1, x_i)$ и в T_2 – $F(T_2, x_i)$ >>.

Простейший набор мер сходства, упорядоченный по возрастанию l имеет вид:

$$q_l(T_1, T_2) = \frac{M_l(T_1, T_2)}{M_l(T_1) + M_l(T_2) - M_l(T_1, T_2)},$$

$l = 1,2,\ldots,l_{\max}(T_1,T_2)$

б) более сложный вариант, учитывающий частоты встречаемости *l*-грамм:

$$\lambda(T_1, T_2) = \frac{\sum_{\alpha} \min\{F(T_1, \alpha), F(T_2, \alpha)\} \cdot |\alpha|}{\sum_{\alpha} \max\{F(T_1, \alpha), F(T_2, \alpha)\} \cdot |\alpha|}$$

где α – произвольная цепочка текстов T_1 и (или) T_2,

$|\alpha|$ – ее длина.

(Findler N.V., Van Leeuwen, 1979)

в) пример расстояния (Kohonen T., Reyhkala E., 1978)

$$h(T_1, T_2) = \max(m(T_1), m(T_2)) - m_0(T_1, T_2) \qquad (*)$$

где $m(T_1)$и $m(T_2)$ – количество признаков, выделяемых из текстов T_1 и T_2, а $m_0(T_1, T_2)$ — число совпавших признаков. Если в качестве признаков использовать l-граммы, получаемые сдвигом вдоль текста скользящего окна размера l, то $m(T_1) = |T_1| - l + 1$, $m(T_2) = |T_2| - l + 1$, и l-граммный аналог (*) имеет вид: .

$$h_l(T_1, T_2) = \max(|T_1| - l + 1, |T_2| - l + 1) - \sum_{i=1}^{M_l(T_1, T_2)} \min\{F(T_1, x_i), F(T_2, x_i)\}$$

г) ранговая мера близости:

Пусть l-граммы в $\Phi_l(T_1)$ и $\Phi_l(T_2)$ упорядочены по убыванию частот; порядковое место l-граммы x_i в упорядочении определяет ее ранг – $r(T_1, x_i)$ (соответственно, $r(T_2, x_i)$).

Группы равночастотных l-грамм представляются усредненным рангом.

Введем l-граммный аналог расстояния:

$$S_l(T_1, T_2) = \sum_{x_i \in \Sigma_l} \left(r(T_1, x_i) - r(T_2, x_i) \right)^2$$

где Σ_l – совокупность всевозможных цепочек длины l ; $|\Sigma_l| = R_l = n^l$.

Аналогом коэффициента Спирмэна для характеристики l-го порядка ($l = 1,2,\ldots$) является

$$\rho_l(T_1, T_2) = 1 - \frac{6 S_l(T_1, T_2)}{R_l(R_l^2 - 1)} \qquad (**)$$

При наличии равночастотных l-грамм в (**) вносится поправка на "связанность" рангов. (Кендел М. Ранговые корреляции, М., Статистика, 1975)