# Исследование повторов в текстах. Алгоритмы

*Гусев Владимир Дмитриевич, к.т.н., ИМ СО РАН*

Кафедра информационной биологии ФЕН НГУ

## Содержание лекции 2.

## 6. Алгоритмы отыскания совершенных повторов.

### 6.1. Основные характеристики алгоритмов – время выполнения T (или временная сложность) и затраты памяти (П) (емкостная сложность).

Они зависят главным образом от размера исходных данных (применительно к текстам – это длина текста $N$; существенным параметром является и размер алфавита $|\Sigma|$).

Оценка трудоемкости и памяти ведется с использованием O-символики.

Утверждение типа "время выполнения $T(N)$ составляет $O(N^2)$" подразумевает, что существуют положительные константы $C$ и $N_0$, такие, что для всех $N$, больших или равных $N_0$, имеет место $T(N) \le C N^2$

Пример. Функция $T(N) = 3N^3 + 2N^2 + 1$ имеет порядок (или степень роста ) $N^3$, т.е. $T(N)$ есть $O(N^3)$.

**Следует иметь в виду, что бóльшему порядку роста одной функции может соответствовать существенно меньшая мультипликативная постоянная, чем у другой функции с малым порядком роста. По этой причине алгоритм с быстро растущей сложностью может оказаться предпочтительнее для задач с малым размером, чем алгоритм с меньшим порядком роста.**

Пример. Зависимость $T(N)$ от размера входа $N$ при разных порядках роста и мультипликативных константах

| Алгоритм | Временная сложность | Диапазон предпочтительных значений $N$ для $A_k$ $(k = 1 \div 4)$ |
|---|---|---|
| $A_1$ | $1000N$ | $N > 1024$ |
| $A_2$ | $100N \log N$ | $59 \le N \le 1024$ |
| $A_3$ | $10N^2$ | $10 \le N \le 58$ |
| $A_4$ | $2^N$ | $2 \le N \le 9$ |

6.2. Метод лексикографической сортировки позволяет вычислить частотную характеристику $l$-го порядка $\Phi_l(T)$ с временными затратами $O(l \cdot N)$ (в предположении, что размер алфавита $|\Sigma| << N$. Затраты памяти — $O(l \cdot N)$)

6.3. Метод, основанный на хешировании символьных цепочек позволяет вычислить $\Phi_l(T)$ с временными затратами

$O(l \cdot N)$ в среднем и затратами памяти $O(N \cdot log\, N)$). Временная сложность может быть уменьшена до $O(N)$ путем построения рекуррентных формул пересчета функции расстановки при сдвиге окна анализа ширины $l$ на одну позицию ("рекуррентное хеширование").

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 $x_i$ (в частности, счетчик, фиксирующий частоту появления $x_i$ в $T$.)

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:

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

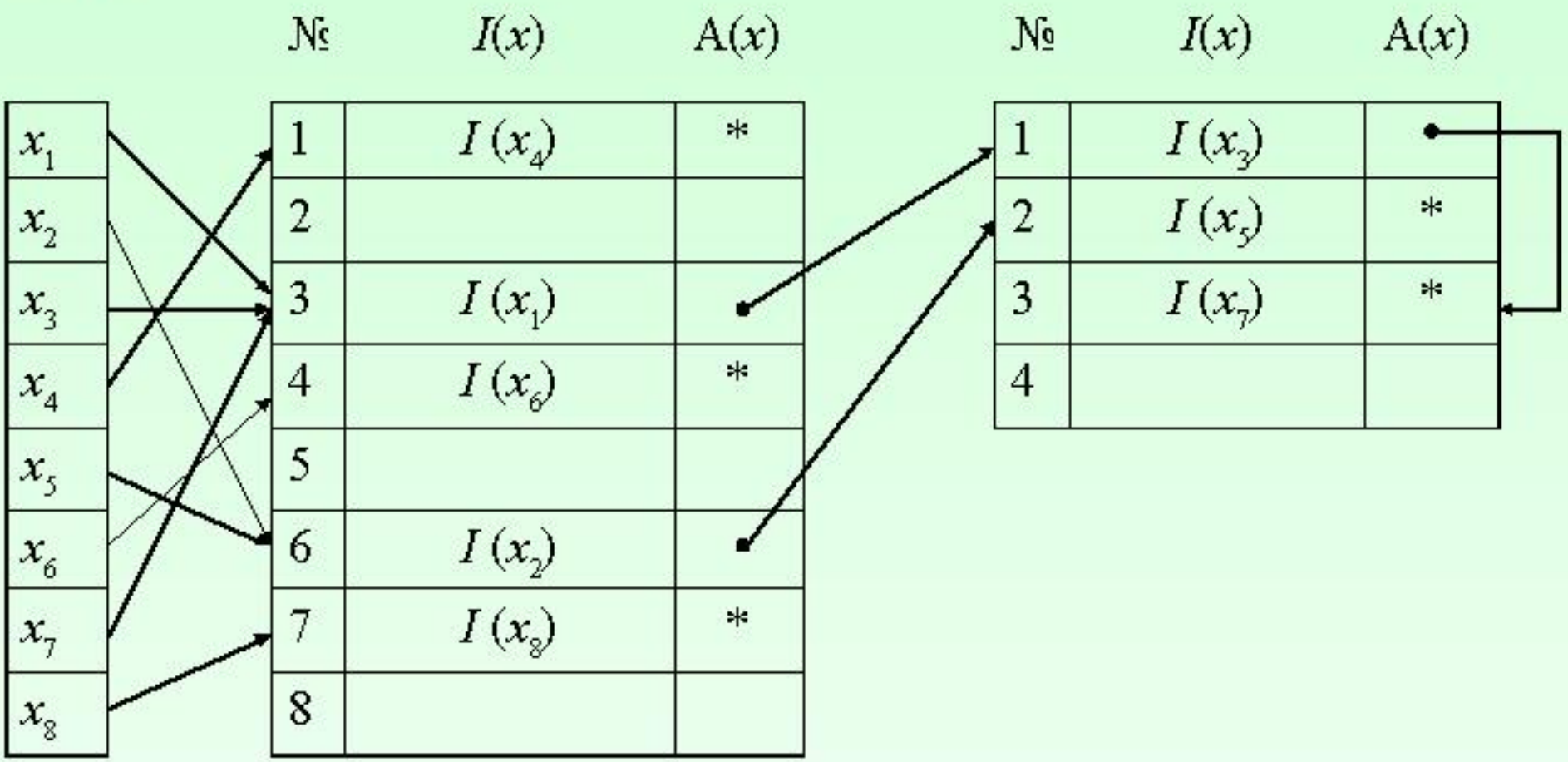
где $K(x_i)$ числовой код $l$-граммы $x_i$,

$M$ — 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



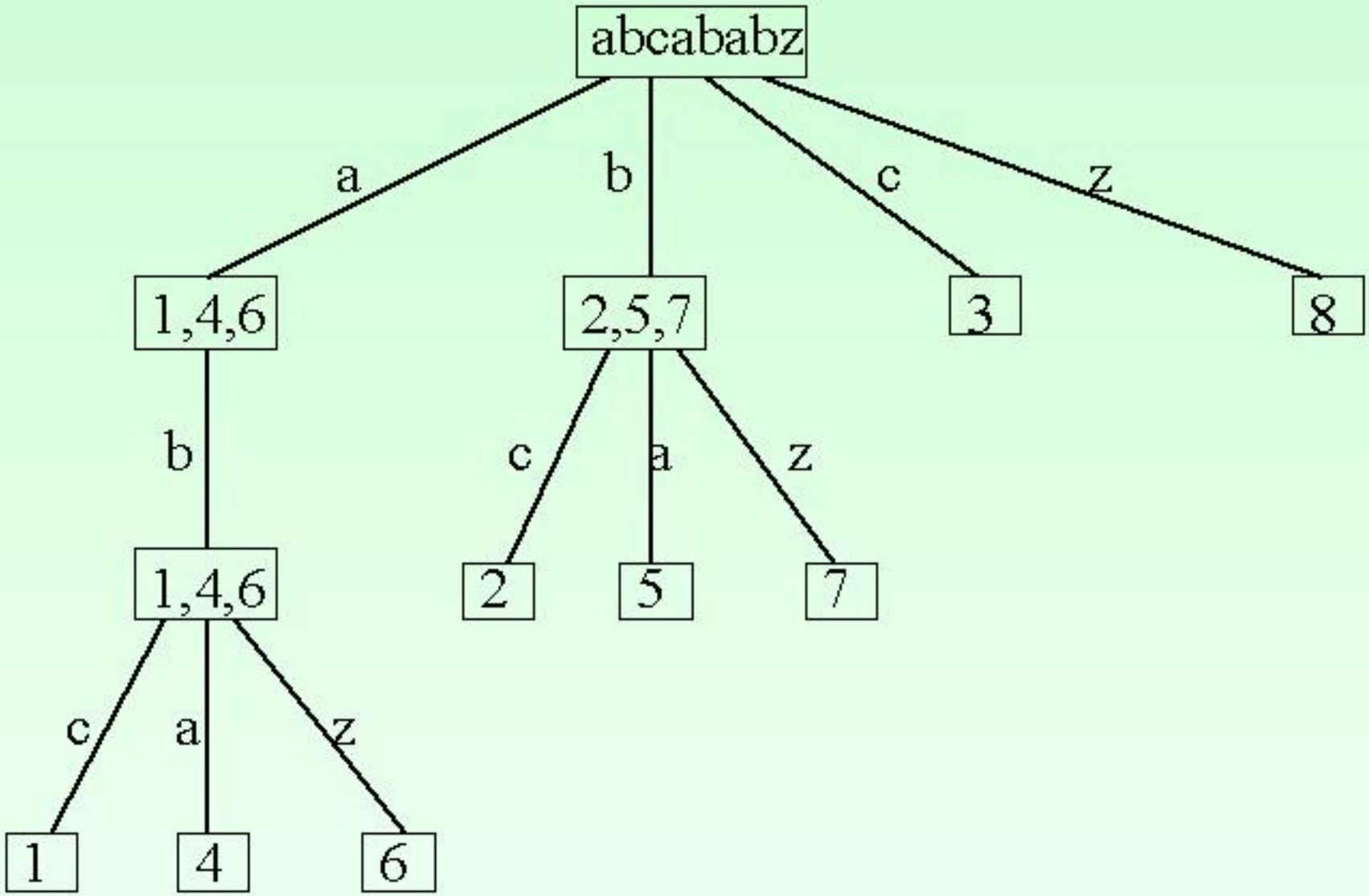
Здесь $x = (x_1, x_2, \ldots x_8)$ — информационный массив, элементы которого хешируются в порядке их следования. Каждый элемент расстановочного поля делится на две части: информационную $I(x)$ и адресную $A(x)$. В $I(x)$ заносится информация о самом первом объекте, распределенном по данному адресу (например, 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.

6.4. Алгоритм Мартинеца позволяет вычислить полный частотный спектр повторов с временными затратами $O(N \log N)$ в среднем и $O(N^2)$ в наихудшем случае. Алгоритм строит дерево из повторяющихся цепочек, склеивая их по общим начальным участкам. Проиллюстрируем процесс построения на примере последовательности Tz=abcababz (здесь z – концевой маркер, $z \notin \Sigma = \{a,b,c\}$).

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 последовательности Tz=abcababz (здесь z – концевой маркер, $z \notin \Sigma = \{a,b,c\}$).



## 6.5. Оптимальные (линейные) алгоритмы вычисления полного частотного спектра.

Если цепочка символов $v$ имеет вид $xyz$, т.е. представлена в виде конкатенации (сцепления) трех цепочек $(x, y, z)$, то $x$ называют префиксом $v$, $z$ – суффиксом, а $y$ – подсловом (подцепочкой). Соответственно, оптимальные алгоритмы отыскания совершенных повторов могут быть основаны на построении префиксного дерева, суффиксного дерева или графа всех подслов текста.

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. Префиксное и суффиксное дерево строятся в режиме "off-line", т.е. требуют сразу предъявления всего текста; граф подслов (или цепочек) строится в режиме "on-line", т.е. по мере поступления новых символов. Рассмотрим для иллюстрации конструкцию **префиксного дерева**.

Префикс–идентификатором $t(i)$ позиции $i$ в тексте $T$ назовем кратчайшую цепочку, начинающуюся в позиции $i$ и встречающуюся в $T$ только один раз.

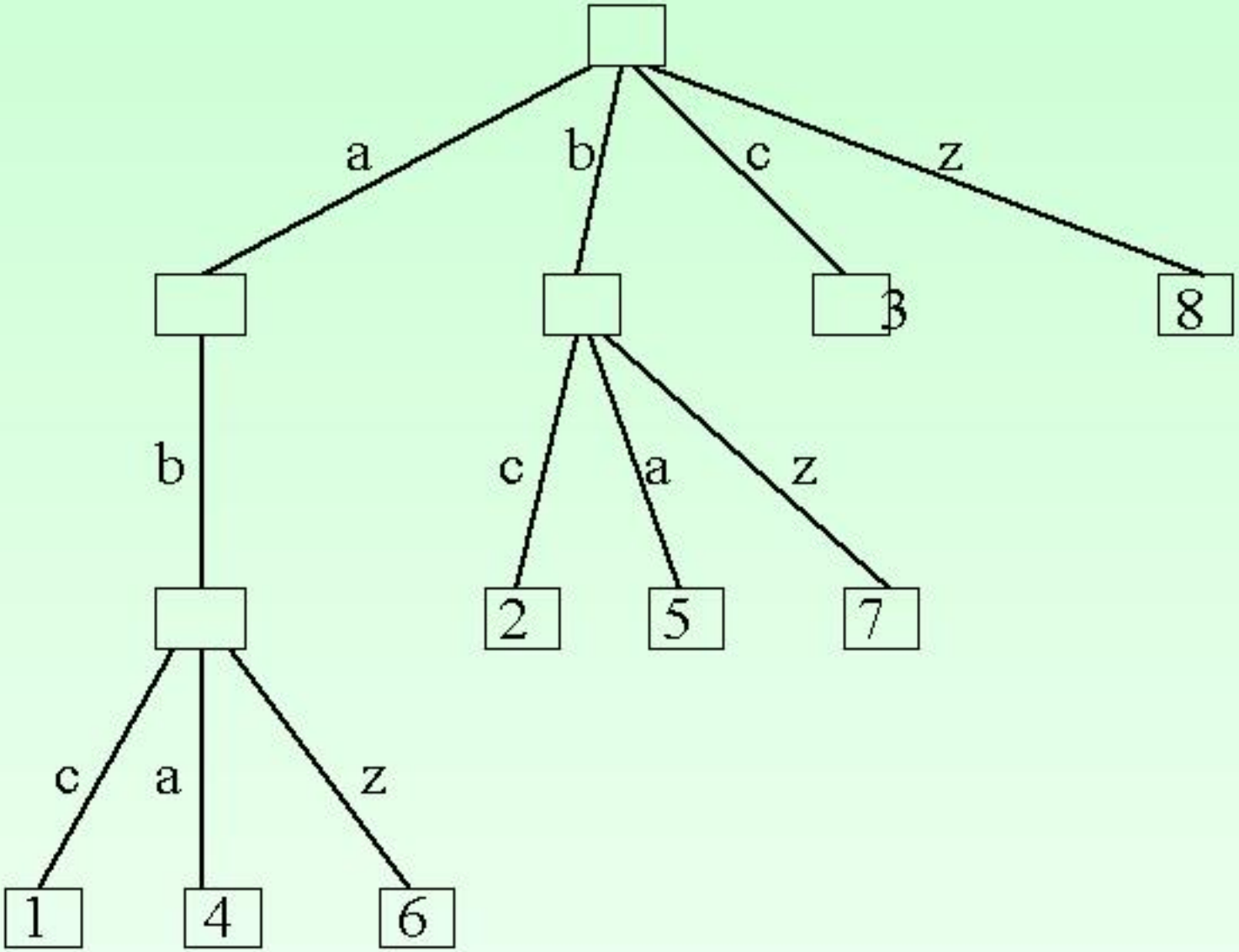
Чтобы $t(i)$ был определен для заключительных позиций текста, его (текст) дополняют конечным маркером $z = a_{N+1}$, который не принадлежит алфавиту $\Sigma$.

Множество всех префикс-идентификаторов можно представить в виде ***префиксного дерева***. Ребра его помечены символами из $(\Sigma \cup z)$, а листья – числами $1,2,\ldots,N+1$, однозначно соответствующими позициям в $Tz$.

Префикс-идентификатор $t(i)$ определяется последовательностью реберных меток на пути из корня в лист с меткой $i$.

# Пример дерева префикс-идентификаторов для цепочки Tz=abcababz

| $i$ | $t(i)$ |
|---|---|
| 1 | abc |
| 2 | bc |
| 3 | c |
| 4 | aba |
| 5 | ba |
| 6 | abz |
| 7 | bz |
| 8 | z |



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 вьше 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.

## 7. Алгоритмы отыскания несовершенных повторов.

Основная трудность: техника, разработанная для отыскания совершенных повторов, имеет очень ограниченное применение для поиска несовершенных повторов.

Пример. Назовем ***(l,k)-повтором*** пару фрагментов текста длины $l$, отличающихся друг от друга точно по $k$ позициям ($k << l$).

Рассмотрим возможности сведения задачи отыскания несовершенных ($l$,$k$)-повторов к отысканию совершенных повторов.

Первая возможность. Пусть для определенности $k = 1$ и размер алфавита n = | Σ | невелик. При $k = 1$ две $l$-граммы, образующие ($l$,1)-повтор, будут иметь лишь пару несовпавших символов в одной из $l$ позиций. Существует $C_n^2$ возможных типов несовпадений (например, для ДНК–алфавита их 6:

A и G, A и C, A и T, G и T, G и C, T и C).

Рассмотрим $C_n^2$ вариантов агрегирования алфавита, "склеивающих" (делающих неразличимыми) пару элементов алфавита, не затрагивая других.

Применительно к ДНК-последовательностям это эквивалентно переходу к следующим 6 алфавитам из 3 элементов каждый:

A=G,C,T; A=T,G,C; A=C,G,T; G=T,A,C; G=C,A,T; T=C,A,G.

Тем самым задача отыскания всех ($l$,1)-повторов в тексте $T$, составленном из элементов алфавита размера $n$, сводится к решению $C_n^2$ задач отыскания совершенных ($l$,0)-повторов в текстах той же длины, но с алфавитом размера $n - 1$. Среди выявленных идеальных повторов отбираются лишь те, которые содержат *одну пару* несовпадающих символов (в исходном алфавите).

Нетрудно видеть, что с ростом параметра $k$ (число допустимых несовпадений) метод быстро становится неэффективным

(в частности, при $k = 3$ и $n = 4$ не работает уже сама идея агрегирования, поскольку две $l$-граммы с тремя несовпадениями типа A,T; T,G; G,C могут быть преобразованы в идеальный повтор лишь перекодировкой вида A = T = C = G при которой тождественными окажутся все $l$-граммы текста).

## Вторая возможность (алгоритмы "ядерного" типа).

Рассмотрим всевозможные варианты расстановки ***k*** несовпадений по ***l*** позициям в паре цепочек, образующих $(l,k)$-повтор. Даже при наихудшем (равномерном) варианте расстановки гарантируется наличие хотя бы одного неискаженного фрагмента ("ядра") размером $l_0 = \left\lceil \frac{l-k}{k+1} \right\rceil$, где $\lceil x \rceil$ означает целое ближайшее к $x$ сверху.

Так, к примеру, при $l = 9$, $k = 2$ гарантированный размер неискаженного ядра равен 3.

Отсюда вытекает ***идея алгоритма***. По заданным ***l*** и ***k*** :

1) вычисляем размер ядра $l_0$;

2) находим в $T$ все совершенные повторы длины $l_0$;

3) расширяем по тексту каждую пару одинаковых ядер до размера $l$-слова в обе стороны (всего существует $l - l_0 + 1$ допустимых вариантов расширения);

4) проверяем каждый вариант на число несовпадений, чтобы не превысить допустимый порог ***k***.

**Эффективность ядерного алгоритма тем выше, чем меньше отношение $k/l$.**

**Однако искусственное завышение размера ядра приводит к потере**

**В общем случае** поиск участков локального сходства (локальной гомологии) осуществляется с помощью процедур **динамического программирования** (соответствующие подходы будут рассмотрены при обсуждении техники выравнивания последовательностей).

Основной недостаток этих методов: временные затраты и память квадратичным образом зависят от длины текста. Возможности ускорения связаны с опасностью потери части допустимых повторов.

Общий вывод: сколь-нибудь развитой классификации несовершенных повторов, по-видимому, не существует. Во многом это связано с тем, что интуитивная трактовка несовершенного повтора как "пары фрагментов близких в определенном смысле" зависит от используемой метрики (или меры близости), которых предложено очень много. Их сравнительный анализ, равно как и выделение отдельных "предметно-ориентированных" классов повторов, а также разработка эффективных алгоритмов их поиска представляют несомненный практический интерес.

# 8. Сложность символьных последовательностей.

## 8.1. Общий подход.

По мнению философов "**сложность** – это общенаучное понятие, приближающееся по своему статусу к философской категории". Применительно к символьной последовательности (тексту) плодотворной оказалась идея А.Н. Колмогорова (1965г) об оценивании ее сложности **длиной кратчайшей программы**, по которой эта последовательность может быть синтезирована. Если последовательность проста, т.е. обнаруживает какие-то регулярные закономерности (например, типа периодичностей) программа ее генерации может быть очень короткой. И, наоборот, если последовательность не обнаруживает каких-либо регулярных закономерностей, кратчайший способ ее генерации может свестись к выписыванию последовательности в явном виде.

Существует множество определений сложности (**энтропийные** меры сложности, **комбинаторная, аддитивная, грамматическая** сложность и др.) Наибольшую популярность получила мера сложности, предложенная **Лемпелем и Зивом** (1976г.), на основе которой реализованы многие программы сжатия текстов. Нас будет интересовать не столько сжатие, как таковое, сколько **выявление интегральных и локальных закономерностей**, на которых оно основано.

## 8.2. Сложность по Лемпелю и Зиву.

В соответствии со схемой Колмогорова Лемпель и Зив измеряют сложность последовательности числом шагов порождающего ее процесса. Допустимыми операциями при этом являются: генерация нового символа и копирование "готового" фрагмента из уже синтезированной части текста. Схема порождения последовательности $S$ по Лемпелю и Зиву (или **сложностное разложение $S$**) может быть представлена в виде конкатенации фрагментов

$$H(S) = S[1:i_1]\,S[i_1+1:i_2]\ldots S[i_{k-1}+1:i_k]\ldots S[i_{m-1}+1:N]$$

$$S[i_{k-1}+1:i_k]$$

где $= m_H(S)$ — фрагмент, синтезируемый на $k$-м шаге,

— число шагов процесса.

Из всевозможных схем порождения $S$ выбирается минимальная по числу шагов. Таким образом сложность последовательности $S$ по Лемпелю и Зиву:

$$K_{LZ}(S) = \min_H\left(m_H(S)\right)$$

**Минимальность числа шагов обеспечивается выбором для копирования на каждом шаге максимально длинного прототипа из предыстории.**

Компоненты $H(S)$ трактуются как "словарь" $S$. Чтобы в нем не было повторяющихся цепочек, найденный максимальный прототип удлиняется еще на один элемент (генерация символа). Таким образом, длина копируемого фрагмента на $k$-м шаге

$$i_k - i_{k-1} - 1 = \max_{j \le i_{k-1}} \left\{ l_j : S[i_{k-1}+1 : i_{k-1}+l_j] = S[j : j+l_j-1] \right\}$$

а сам $k$-й компонент

$$S[i_{k-1}+1 : i_k] = \begin{cases} S[j(k) : j(k)+l_{j(k)}-1]\, S[i_k] & \text{при} \quad j(k) \neq 0, \\ S[i_{k-1}+1] & \text{при} \quad j(k) = 0, \end{cases}$$

где $j(k)$— номер позиции, с которой начинается копирование на $k$-м шаге ($j(k) = 0$, если в позиции $i_{k-1}+1$ стоит ранее не встречавшийся символ),

$l_{j(k)}$ — длина копируемого фрагмента.

Пример 1. Пусть $\Sigma = \{A,B\}$ и $S = ABBABAABBAABABBA$.

Схема порождения $S$ имеет вид: $H(S) =$ A·B·B·BA·BAA·BBAA·BABB·A ,

$C_{LZ} = 7$.

Здесь компоненты разложения отделены друг от друга точками, а копируемые фрагменты подчеркнуты снизу.

## 8.3. Модификации меры Лемпеля и Зива.

Мера $C_1$ : к копируемому фрагменту не добавляется на каждом шаге еще один символ, т.е. операция генерации символа используется только для введения в текст не встречавшихся ранее элементов алфавита. Как следствие, в сложностном разложении фигурируют теперь "чистые" повторы.

Пример 2. Для рассмотренной в примере 1 последовательности $S$ получаем следующее разложение: $H_1(S) = \text{A}\cdot\text{B}\cdot\text{B}\cdot\text{AB}\cdot\text{A}\cdot\text{ABBA}\cdot\text{ABA}\cdot\text{BBA}$, $C_1(S) = 8$.

Заметим, что выбор указателя копирования для одного и того же компонента разложения может быть неоднозначным. Так, последний (восьмой) компонент BBA допускает два варианта копирования со значениями $j(8) = 2$ и $j(8) = 8$. При выявлении структурных особенностей текста удобнее использовать ***наибольший*** из возможных указателей копирования. При этом копируемый фрагмент и его прототип оказываются максимально сближенными, что иногда свидетельствует об их совместном функционировании. Мы будем придерживаться последней из указанных стратегий.

## Два полезных свойства меры $C_1$.

**Свойство 1. Расширяемость компонентов.** Компонент разложения и его прототип нерасширяемы вправо, но могут допускать расширение влево, трактуемое как удлинение повтора. Число добавляемых слева символов при этом строго меньше длины предыдущего компонента.

**Пример 3.** Пусть $\Sigma = \{A,C,G,T\}$ и $\boldsymbol{S}$ = ATAAGCTTAATTAAGCTTC.

Схема порождения $S$ имеет вид:

$\boldsymbol{H_1(S)}$ = A·T·A·A·G·C·T·T<u>AA·TTAA·GCTT</u>·C, $\boldsymbol{C_1(S)} = 11$.

Здесь прототипом для предпоследнего (10-го) компонента является цепочка $S$ [5 : 8]. Нетрудно видеть, что 10-й компонент и его прототип могут быть расширены влево на 3 символа, образуя более длинный повтор (см. подчеркнутые цепочки). Это означает, что мы имеем для $S$ эквивалентные (по числу компонентов) разложения — приведенное выше и $H'_1(\boldsymbol{S})$ = A·T·A·A·G·C·T·T<u>AA·T·TAAGCTT</u>·C, $\boldsymbol{C_1}(S) = 11$. В $H'_1(\boldsymbol{S})$ 9-й компонент редуцирован, т.е. получен не по оптимальной схеме, а 10-й удлинен ($j(10) = 2$).

# Свойство 2. Обнаружение тандемных повторов.

Пусть текущий ($k$-й) порождаемый компонент начинается в позиции $q_k$, его длина — $l_k$, указатель копирования — $j(k)$. Если $j(k)+ l_k \geq q_k$, т.е. прототип вплотную примыкает к порождаемому компоненту или накладывается на него, то имеет место тандемная повторность с длиной периода $t = q_k - j(k)$ и кратностью повторений не меньшей, чем $entier\ (l_k / t) + 1$.

**Пример 4**. Схема порождения $S$ имеет вид

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $H_1(\boldsymbol{S})$ = | G· | T· | A· | TA· | C· | CC· | G· | GTATA· | | | | | | | TATATA· | | | A· |
| № комп. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | | | | | | 9 | | | 10 |
| $j(k)$ | 0 | 0 | 0 | 2 | 0 | 6 | 1 | 1 | | | | | | | 13 | | | 20 |

Здесь компоненты №№ 4 (TA), 6 (CC), 9 ($(TA)^3$) копируются с прототипов, непосредственно примыкающих к ним (№ 4) или налагающихся на них (№ 6 и № 9). Так, прототипом для компонента № 9 ($(TA)^3$), начинающегося в позиции $q_9 = 15$, является цепочка $(TA)^3$, начинающаяся в позиции $j(9) = 13$. При этом длина периода $t = q_9 - j(9) = 15 - 13 = 2$, сам период — цепочка TA, кратность повторений — не меньше, чем $entier\ (6 / 2) + 1 = 4$ . Реальная кратность равна 5.

Мера $C_2$— ДНК-ориентированный вариант меры сложности $C_1$. Сложностное разложение текста и значение сложности зависят от типа используемой операции копирования. Так, если в мере $C_1$ заменить операцию прямого копирования на симметричное копирование, получим для S из примера 2 следующее разложение:

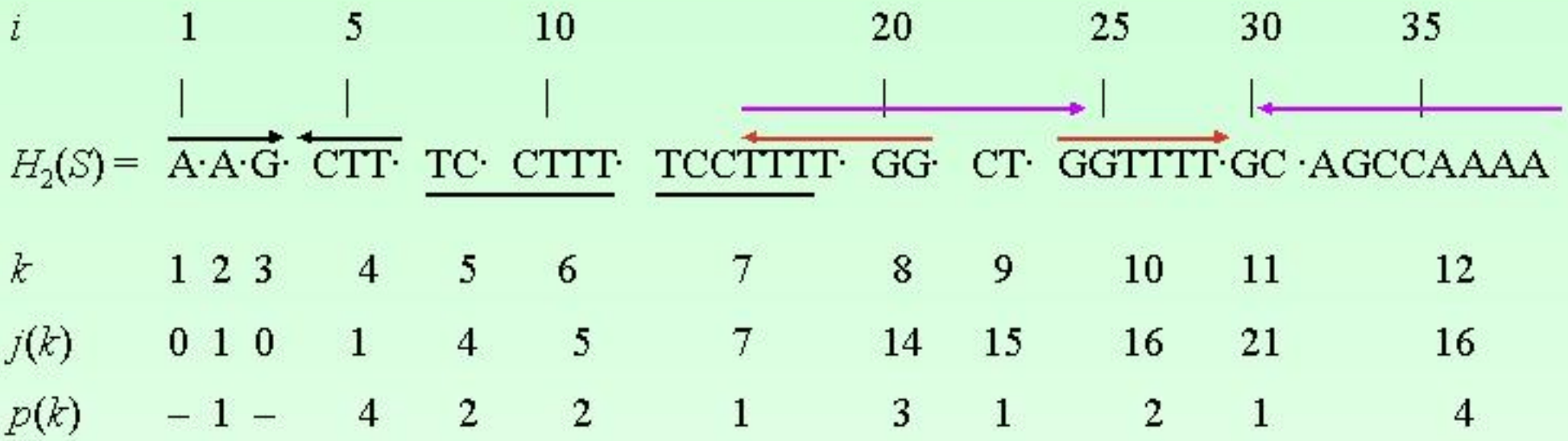
$$H_1^{\text{сим}}(S) = \text{A}\cdot\text{B}\cdot\text{BA}\cdot\text{BA}\cdot\text{A BBA}\cdot\text{ABABBA}, \quad C_1^{\text{сим}}(\text{S}) = 6,$$

т.е. сложность уменьшилась на 2.

Отсюда возникает следующая возможность обобщения меры $C_1$:

введем 4 операции копирования (прямое (1), симметричное (2), прямое комплементарное (3) и симметричное комплементарное (4)) в соответствии с 4 выделенными ранее типами повторов и будем выбирать на каждом шаге генерации текста ту операцию копирования, которой соответствует максимальный прототип в предыстории. Получим меру $C_2$. Каждый компонент разложения в ней снабжается (кроме указателя копирования) указателем способа копирования (или типа повтора) $p(k)$.

Пример разложения некодирующей последовательности $S$ из генома дрожжей по мере $C_2$. Схема порождения $S$ имеет вид:




| $k$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $j(k)$ | 0 | 1 | 0 | 1 | 4 | 5 | 7 | 14 | 15 | 16 | 21 | 16 |
| $p(k)$ | – | 1 | – | 4 | 2 | 2 | 1 | 3 | 1 | 2 | 1 | 4 |

$$C_2(S) = k_{\max} = 12.$$

Выделены наиболее характерные структуры: подчеркнут тандемный повтор, стрелками сверху отмечены симметричный повтор (← →) и симметричный комплементарный повтор (→ ←), образующий структуру шпилечного типа с симметричной петлей GTTTTG.

## 8.4. Режимы использования мер $C_1$ и $C_2$. Возможные обобщения.

1) Число возможных подстановок (переименований элементов алфавита) составляет $n!$, где $n = |\Sigma|$. С учетом прямого и симметричного копирования можно зафиксировать 2 $n!$ различных операций копирования и вычислить вектор из 2 $n!$ значений сложности $C_1$ (**векторная мера сложности** – еще один способ представления последовательностей).
2) Можно сформулировать меру $C_3$ (уже не являющуюся предметно-ориентированной в отличие от $C_2$), где на каждом шаге выбирается та из $2n!$ допустимых операций копирования, которой соответствует максимальный прототип в предыстории (алгоритм реализован).
3) Вычисление мер $C_1$ и $C_2$ в скользящем окне размера D позволяет построить **сложностной профиль** последовательности и выявить фрагменты с аномально низкой сложностью, характеризующиеся наиболее регулярной структурой. Они часто оказываются функционально значимыми. Параметр D характеризует разрешающую способность анализа.
4) Можно получать сложностное разложение одной последовательности по другой, т.е. представлять одну последовательность в виде конкатенации фрагментов из другой. Эту процедуру можно распространить на группу текстов, что позволяет за один просмотр подборки выявить наиболее существенные взаимосвязи между текстами в виде повторов разного типа.

# Вопросы, упражнения, задачи.

1. Сформулируйте алгоритм случайного перемешивания текста с сохранением его биграммного состава.
2. Пусть в модульной функции расстановки $h(x) = x \bmod N$ в качестве параметра $N$ фигурирует четное число. Приведите пример информационного массива, состоящего из набора чисел $x_1, x_2, \ldots x_m$, для которых такой выбор $N$ оказался бы неудачным.
3. Постройте дерево префикс-идентификаторов для цепочек $S_1$ = baaaab и $S_2$ = abababa в двух вариантах : а) некомпрессированное; б) компактное.
4. Покажите, что: а) если префикс-идентификатор $t(i)$ имеет длину $j$, то длина $t(i-1)$ не превосходит $j+1$; б) никакой префикс-идентификатор не является собственным префиксом другого.
5. Покажите, что позиционное дерево для текста $T = a^k b^k a^k b^k z$ содержит $k^2 + 6k + 2$ узлов.

6. Опишите алгоритм поиска цепочки $x = b_1 b_2 \ldots b_p$ в тексте $T = a_1 a_2 \ldots a_n$, для которого уже построено дерево префикс-идентификаторов.

7. Сформулируйте алгоритм отыскания $(l,k)$-повторов ($k << l$) с кластеризованными (расположенными подряд) заменами.

8*. Переформулируйте задачу 6 для случая цепочек, отличающихся друг от друга $k$- элементной блочной вставкой или делецией. Предложите алгоритм отыскания таких повторов.

9*. Предложите алгоритм поиска в тексте фрагментов фиксированной длины с одинаковым $l$-граммным составом.

10. Пусть $\Sigma = \{A,G,C,T\}$ – алфавит нуклеотидов. Напишите цепочку минимальной длины, содержащую: а) все возможные биграммы в данном алфавите; б) все триграммы. Единственное ли решение? Оцените сложность цепочек по мере $C_1$.

11. Приведите пример сложностного разложения по мере $C_2$, содержащего хотя бы один компонент, который может быть получен с использованием разных операций копирования.

12. Получите сложностные разложения по мере $C_1$ и $C_2$ для последовательностей :

а) TGTGTCACAGACACAGA (фрагмент эукариотического промотора)

б) CCAGTCCGTCGCCTGCCA (— // —)

в) TGTAACGAAAATTTCCAATGT (— // —)

Выделите стрелками, подчеркиванием выявляемые структуры.

13. Для последовательности CCAACAAGCAATGTTGG (фрагмент эукариотического промотора) получите 4 типа разложений по мере $C_1$ с использованием, соответственно, одной из 4 операций копирования (прямое, симметричное, прямое комплементарное и симметричное комплементарное).

14. Приведите пример, когда однократная замена в НК-последовательности приводит:

а) к увеличению сложности по мере $C_1$ ;

б) к уменьшению сложности по мере $C_2$ .