Расстояния между строками и меры близости

Расстояние Хэмминга: число несовпадений символов.

Пример: A = abbaeac; $d_{хэм} = 2$.

B = abdaecc;

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

Пример: A = abbaeac; B = bdedac

$A \rightarrow B$:

a	b	b	a	e	a	c
–	b	d	e	d	a	c

$d(A,B) = 4$ (1 дел., 3 зам).

$A \rightarrow B$:

a	b	b	a	e	–	a	c
–	b	d	–	e	d	a	c

$d(A,B) = 4$ (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:

$d(0,0) = 0;$

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

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

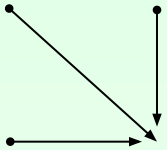
$$d(i,j) = \min \{d(i-1,\ j-1) + \gamma(a_i,b_j),\ d(i-1,\ j) + 1,\ d(i,\ j-1) + 1\}.$$

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

B

	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**

A



- Этап 1: заполнение матрицы динамического программирования.
- Этап 2: обратный ход. Построение выравнивания. $\forall$ $d(i,j)$ на этапе 1 запоминаем указатели на элемент, от которого реализовался переход ($d[i-1, j-1]$, $d[i-1, j]$ и /или $d[i, j-1]$; либо на этапе 2 делаем проверку $d[i, j] = d[i-1, j-1] + \gamma(a_i \rightarrow b_j)$?, или $d[i,j] = d[i-1, j] + \gamma(a_i \rightarrow \varepsilon)$?, или $d[i,j] = d[i, j-1] + \gamma(\varepsilon \rightarrow b_j)$?.

Трудоемкость и затраты памяти $O(n \times m)$.

Если требуется только вычислить редакционное расстояние (без выравнивания) можно хранить две строки: текущую и предыдущую.

Для построения выравнивания требуется знать всю матрицу.

–	**b**	**d**	**–**	**e**	**d**	**a**	**c**	**или**	**–**	**b**	**d**	**e**	**d**	**a**	**c**
a	**b**	**b**	**a**	**e**	**–**	**a**	**c**		**a**	**b**	**b**	**a**	**e**	**a**	**c**

- В общем случае число редакционных операций можно расширить, например, добавить перестановку соседних символов или блочные вставки или устранения.
- Каждая операция может иметь свой "вес". В этом случае редакционное расстояние определяется как минимальная "стоимость" перевода A в B.

- Если к редакционным операциям добавлена операция перестановки соседних символов, схема динамического программирования выглядит следующим образом:

$$d[\,i,\,j\,] = \min\{\, d[i-1,\,j-1] + \gamma(a_i \to b_j),$$
$$d[i-1,\,j\,] + \gamma(a_i \to \varepsilon),$$
$$d[i,\quad j-1] + \gamma(\varepsilon \to b_j)$$
$$d[i-2,\,j-2] + \gamma(a_{i-1}\,a_i \to b_{j-1}\,b_j).\ \text{если}\ a_i\,a_{i-1} = b_{j-1}\,b_j\}$$

при тех же начальных условиях

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

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

Возможные обобщения

- Поиск участков текста, **близких к заданному образцу**.
- Может формулироваться в виде: найти все фрагменты текста, для которых редакционное расстояние с образцом не превышает ***R*** (некоторый заданный параметр).
- Для решения этой задачи достаточно стоить матрицу ***D***, изменив начальные условия, а именно, положить $d[0, j] = 0$, для всех $1 \le j \le n$.
- Пример. P = baaa; T = bbabbaabab;

	ε	b	b	a	b	b	a	a	b	a	b
ε	0	0	0	0	0	0	0	0	0	0	0
b	1	0	0	1	0	0	1	1	0	1	0
a	2	1	1	0	1	1	0	1	1	0	1
a	3	2	2	1	1	2	1	0	1	1	1
a	4	3	3	2	2	2	2	1	1	1	2

Возможные обобщения:

- Поиск «похожих» фрагментов текста.
- Схема динамического программирования, но с начальными условиями:

$d[0, j] = 0$, для $0 \le j \le n$ и $d[i, 0] = 0$ для $0 \le i \le m$.

- Однако удобнее использовать "меру близости", т.е. величину, противоположную ред. расстоянию: чем тексты ближе, тем бо́льшие значения имеет мера близости.

Мера близости определяется следующими соотношениями

$$r(i, j) = \max \{ \; 0,$$
$$r(i-1, j-1) + \delta(a_i, b_j),$$
$$r(i-1, j) + \delta(a_i, \varepsilon),$$
$$r(i, j-1) + \delta(\varepsilon, b_j)\}$$

$\delta(a_i, b_j) = 1$, если $a_i = b_j$; $\quad \delta(a_i, \varepsilon) = \delta(\varepsilon, b_j) = -2$.

$\delta(a_i, b_j) = -2$, если $a_i \ne b_j$.

$r[0, j] = 0$, $r[i, 0] = 0$, $\quad 0 \le j \le n$, $0 \le i \le m$.

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

$$r(i,j) = \max\{0,$$
$$r(i-1,j-1) + \delta(a_i, b_j),$$
$$r(i-1,j) + \delta(a_i, \varepsilon),$$
$$r(i,j-1) + \delta(\varepsilon, b_j)\}$$

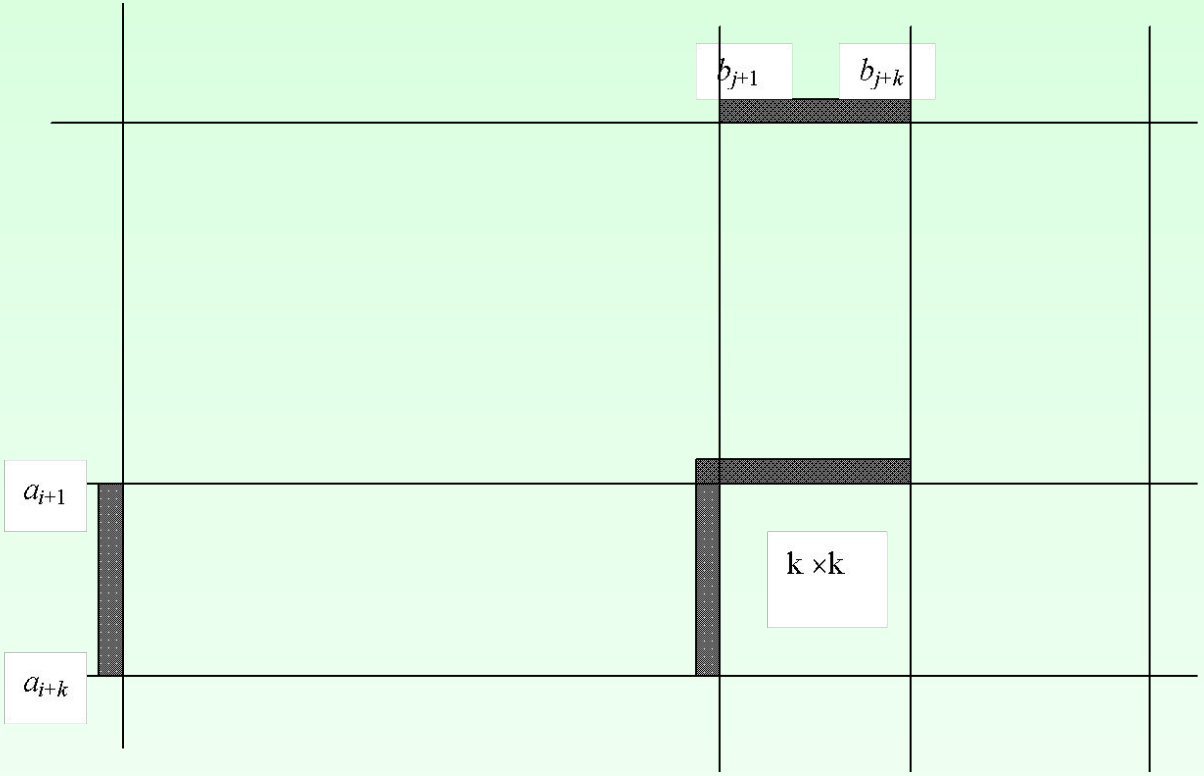
$\delta(a_i, b_j) = 1$, если $a_i = b_j$
$\delta(a_i, b_j) = -2$, если $a_i \neq b_j$
$\delta(a_i, \varepsilon) = \delta(\varepsilon, b_j) = -2.$

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

$r'(i, n+1) = 0;$
$r'(m+1, j) = 0;$

$$r'(i,j) = \max\{0,$$
$$r'(i+1,j+1) + \delta(a_i, b_j),$$
$$r'(i+1,j) + \delta(a_i, \varepsilon),$$
$$r'(i,j+1) + \delta(\varepsilon, b_j)\}$$

Алгоритм Мазека-Патерсона. Для вычисления подматрицы D размера k × k должны быть известны начальные векторы и соответствующие фрагменты исходных текстов. Идея четырех русских: Арлазаров, Диниц, Кронрод, Фараджиев. Ускорение достигается за счет предварительного вычисления и сохранения информации обо всех вариантах вызова подзадачи. Тогда при решении задачи можно по введенным аргументам (подстроки текста и начальные векторы) сразу выписать ответ: конечные векторы, которые, в свою очередь, будут начальными для следующих фрагментов. Для сокращения числа аргументов можно кодировать разности между соседними элементами. $O(n^2/\log n)$



Максимально длинная общая подпоследовательность (МДП, LCS)

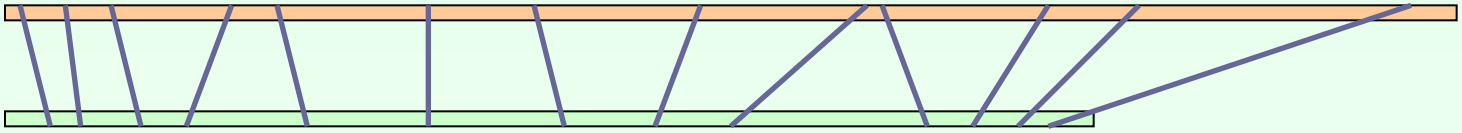
Последовательность U является подпоследовательностью A, если существует монотонно возрастающая последовательность целых чисел $r_1 \ldots r_{|U|}$ такая, что $U[j] = A[r_j]$, $1 \le j \le |U|$, $1 \le r_j \le |A|$.

Если $\gamma(a_i \to b_j) \ge \gamma(a_i \to \varepsilon) + \gamma(\varepsilon \to b_j)$, то при переводе A в B используются только операции вставки и устранения, а элементы, составляющие LCS, остаются без изменения.

Если $\gamma(a_i \to b_j) = 2$ (при $a_i \ne b_j$) , а $\gamma(a_i \to \varepsilon) = \gamma(\varepsilon \to b_j) = 1$,

$D(A,B) = m + n - 2L(A,B)$, $m = |A|$, $n = |B|$

$2L(A,B) = m + n - D(A,B)$



Вычисление длины МПД:

$$L(i,j)=\begin{cases} L(i-1,j-1)+1, & \text{если} \quad a_i = b_j \\ \max(L(i,j-1), \quad L(j-1,j)) & \text{в остальных случаях} \end{cases}$$

$L(0, j) = 0; \quad L(i, 0) = 0;$

$0 \le i \le m; \quad 0 \le j \le n;$

Пример. A = aacacbb; B = ababc.
L(A,B) = 3.

Всего 12 МДП:

3 – abb,

6 – aab,

3 – aac

	ε	a	b	a	b	c
ε	0	0	0	0	0	0
a	0	1	1	1	1	1
a	0	1	1	2	2	2
c	0	1	1	2	2	3
a	0	1	1	2	2	3
c	0	1	1	2	2	3
b	0	1	2	2	3	3
b	0	1	2	2	3	3

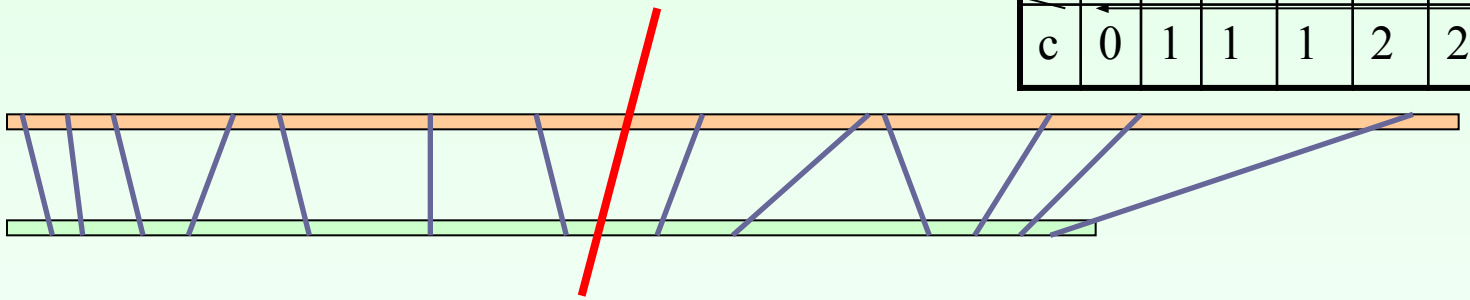
Алгоритм Хишберга (Hirschberg D.S.)

Пусть $L^*(i, j)$ – длина МДП текстов A[$i + 1 : m$] и B[$j + 1 : n$].

Тогда для любого i: $L(m,n) = \max_j \{L(i, j) + L^*(i, j)\}$

A = aaca'cbb	A1 = aa'ca	A2 = cb'b
B = ababc	B1 = aba	B2 = bc
$L(4,0) + L^*(4,0) = 0 + 2$	$L(2,0) + L^*(2,0) = 0 + 1$	$L(2,0) + L^*(2,0) = 0 + 1$
$L(4,1) + L^*(4,1) = 1 + 2$	$L(2,1) + L^*(2,1) = \underline{1 + 1}$	$L(2,1) + L^*(2,1) = \underline{1 + 0}$
$L(4,2) + L^*(4,2) = 1 + 1$	$L(2,2) + L^*(2,2) = 1 + 1$	$L(2,2) + L^*(2,2) = 1 + 0$
$L(4,3) + L^*(4,3) = \underline{2 + 1}$	$L(2,3) + L^*(2,3) = 2 + 0$	
$L(4,4) + L^*(4,4) = 2 + 1$		
$L(4,5) + L^*(4,5) = 3 + 0$		

	ε	c	b	a	c	a
ε	0	0	0	0	0	0
b	0	0	1	1	1	1
b	0	0	1	1	1	1
c	0	1	1	1	2	2



Адаптивные алгоритмы вычисления длины МПД:

Hunt - Shimanski

$Q_{i\,k}$ – наим. j, такое что (L(A[1 : i], B[1 : j]) = k

$$Q_{i+1,k} = \begin{cases} наим.\, j, & такое\ что \quad Q_{i,k-1} < j \le Q_{ik} \quad и \quad a_{i+1} = b_j \\ Q_{ik}, & если \quad такого \quad j \quad не\ существует \end{cases}$$

Начальные условия:

$Q_{i,0} = 0, \qquad 0 \le i \le n;$

$Q_{0k} = n + 1, \qquad 1 \le k \le \min(m,n);$

Пример. A = aacacbb,

B = ababc.

Трудоемкость: O($r \times \log n$),

где $r = \sum_{i=1}^{|\Sigma|} f_i(A) \cdot f_i(B)$

число потенциально возможных парных соответствий символов

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

Адаптивные алгоритмы вычисления длины МПД:

Nakatsu-Kambayashi-Yajima

$R_{i\,k}$ – наиб. j, такое что (L(A[i : m], B[j : n]) = k

$$R_{i,k} = \begin{cases} \textit{наиб. } j, \quad \textit{такое что} \quad R_{i+1,k} \le j < R_{i+1,k-1} \quad \textit{и} \quad a_i = b_j \\ R_{i+1,k,} \qquad\qquad \textit{если такого } j \textit{ не существует} \end{cases}$$

Начальные условия:

$R_{i,0} = n + 1, \quad 0 \le i \le m;$

$R_{m+2-k,\,k} = 0, \quad 1 \le k \le \min(m,n);$

Пример. A = aacacbb,

B = ababc.

Трудоемкость: O(n× (m – L)),

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

- **Близкие задачи:**

- задача о наикратчайшей **надпоследовательности**

- задача о **медиане** (string merging): построение текста T3, сумма переходов к которому от T1 и T2 минимальная. В зависимости от весов ред. операция в качестве T3 можно получить МДП, наименьшую надпоследовательность, один из текстов (T1 или T2), наиболее вероятного предка и т.п.

- поиск максимально длинной общей подпоследовательности для группы текстов.

- задача о максимально длинной возрастающей (убывающей) подпоследовательности для числовой перестановки

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

$$\Phi_l(T_1, T_2) = \{\varphi_{l1}(T_1, T_2), \varphi_{l2}(T_1, T_2), \ldots, \varphi_{l\,Ml}(T_1, T_2)\}$$

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

а элемент $\varphi_{l\,i}$ $(1 \le i \le M_l)$ 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 Leeuten, 1979)

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:

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

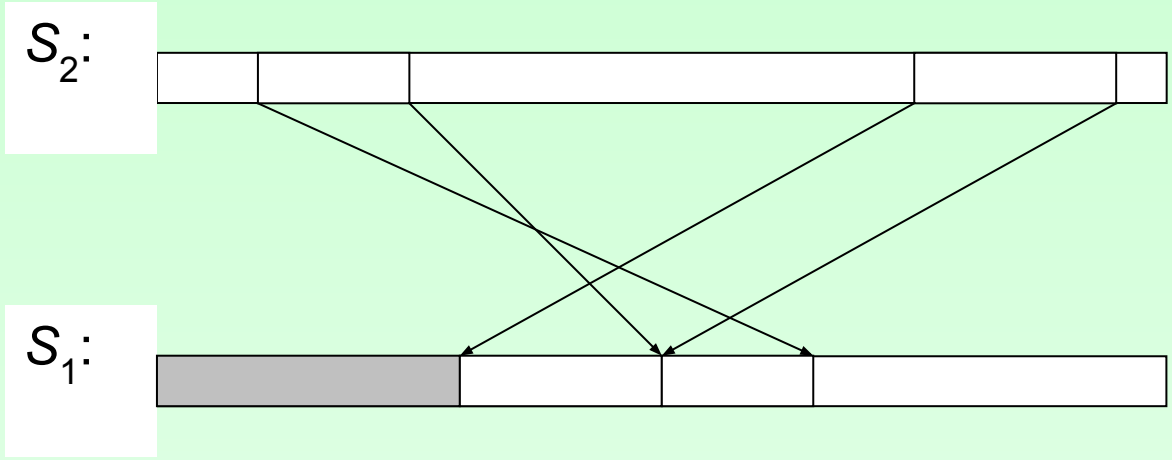
где Σ_l – совокупность всевозможных цепочек длины l ; $|\Sigma_l| = R_l = n^l$.

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

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

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

Обобщение подхода Лемпеля-Зива



- Представление S_1 в виде конкатенации фрагментов из S_2 назовем **сложностным разложением** S_1 по S_2.
- На каждом шаге копируется максимальный фрагмент S_2, совпадающий с префиксом непокрытого участка S_1
- Если такого фрагмента нет, используется операция генерации символа
- **c(S_1 / S_2) – сложность** S_1 относительно S_2 определяется числом компонентов в разложении S_1 по S_2

Относительная сложность и редакционное расстояние

S_2 = aaaa a cccc c ttttttttttttt – acacacac a atatatat

S_1 = aaaa g cccc g ttttttttttttt g acacacac g atatatat

$d(S_1, S_2)$ = 4

$H(S_1/S_2)$ = aaaa*g*cccc*g*ttttttttttttt*g*acacacac*g*atatatat

$c(S_1/S_2)$ = 9

- $c\ (S_1/S_2) \leq 2d(S_1, S_2) + 1$

- S_2 = --------ttttttttttttttttaaaaaaaa

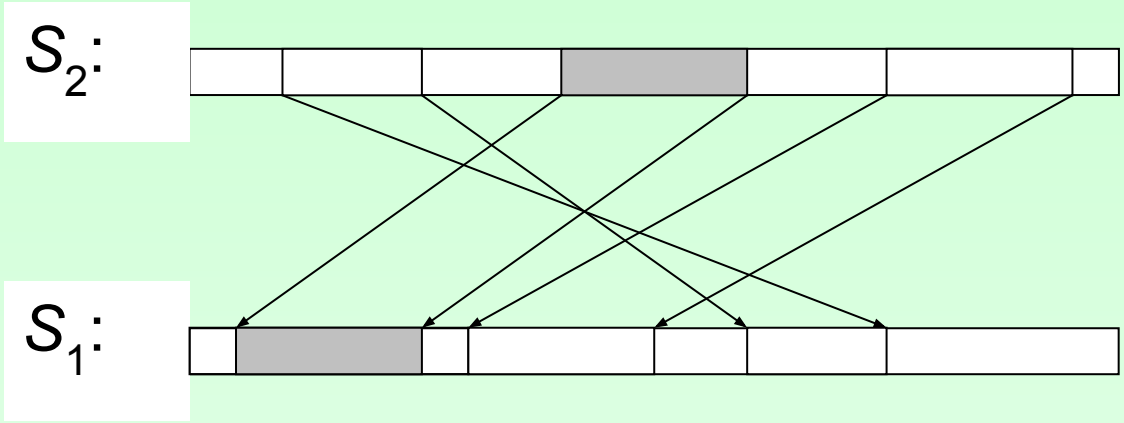
  S_1 = aaaaaaaattttttttttttttttt--------

  $d\ (S_1\ ,\ S_2) = 16$

  $H(S_1\ /\ S_2)$ = aaaaa * ttttttttttttttttt

  $c\ (S_1\ /\ S_2) = 2$

Трансформационное расстояние



- Трансформационое расстояние и относительная сложность идейно близки.
- Операция «вставка сегмента» используется, если посимвольная генерация фрагмента «дешевле» его копирования.
- Порядок покрытия S_1 предполагает оптимизацию по всем парам межтекстовых повторов и промежуткам между ними. $O(N^6)$.

J.-S.Varré, J.-P.Delahaye, E. Rivals: Transformation Distances: a Family of Dissimilarity Measures Based on Movements of Segments. // Bioinformatics 15(3): 194-202 (1999)

Инверсионное расстояние

$$\begin{pmatrix} 1\,2\,\ldots\, i-1\; i\; i+1\,\ldots\, j-1\; j\; j+1\,\ldots\, N \\ 1\,2\,\ldots\, i-1\; j\; j-1\,\ldots\, i+1\; i\; j+1\,\ldots\, N \end{pmatrix}$$

- Инверсионное расстояние $d_I(\pi,\sigma)$ между последовательностями π и σ определяется минимальным числом инверсий, переводящих одну из них в другую
  Задача вычисления инверсионного расстояния для перестановок является NP-полной
- В случае "знаковых" перестановок существуют полиномиальные решения

  +1 [+2 −4 −5] +3 +6 → +1 +5 +4 −2 +3 +6

Hannenhalli, S. and Pevzner, P. Transforming Cabbage into Turnip (Polynomial Algorithm for Sorting Signed Permutation by Reversals). Proc. 27th Ann. ACM Symposium on the Theory of Computing, 1995, pp. 178–189

Точки разрыва

$\pi_0 = 0$ and $\pi_{N+1} = N + 1$

π and σ – произвольные перестановки.

Разрыв между $\pi_i = a$ и $\pi_{i+1} = b$ фиксируется, если в σ нет биграмм ab и ba.

π = 0 | 6 4 | 1 8 5 | 3 | 2 9 7 | 10 $\quad$ $r(\pi, \sigma) = 5$

σ = 0 | 5 8 1 | 2 9 7 | 6 4 | 3 | 10

σ – тождественная перестановка (1 2 … N).

Число точек разрывов (**breakpoint distance**) $r(\pi, \sigma)$ определяется количеством позиций π таких что $|\pi_i - \pi_{i+1}| \neq 1$.

1 2 3 | 8 7 6 | 4 5 | 9

Инверсионное расстояние, число точек разрыва и относительная сложность

- $r(\pi,\sigma) \leq 2d_I(\pi,\sigma)$

- Точки разрыва однозначно соответствуют границам компонентов сложностного разложения
  $r(\pi,\sigma) = c(\pi/\sigma) - 1$
  σ = 1 2 3 4 5 6 7 8 9
  H(π / σ) = 1 2 3 * 8 7 6 * 4 5 * 9

- Сложностные разложения позволяют перейти от исходных перестановок к "знаковым" и сократить размерность задачи вычисления инверсионного расстояния