Алгоритмы на строках

Основные понятия

Строка – это последовательность символов из определенного алфавита

Подстрока строки S определяется как `S[i..j]` для любой пары таких чисел `i` и `j`, что

`1 <= i <= j <= n`,

где n – количество символов в строке S (длина строки S).

Количество символов в подстроке равно `(j – i + 1)`.
Можно выделить `(n – k + 1)` подстрок длины k из строки S длины n. Это подстроки

`S[1..k], S[2..(k+1)], ..., S[(n-k+1)..n]`

Общее количество подстрок с длинами от 1 до n имеет порядок n^2 и равно

$$\sum_{k=1}^{n}(n-k+1)=\frac{n^2+n}{2}$$

Основные определения

Пусть имеются две строки $S_1[1..n]$ и $S_2[1..m]$.

1. Строки S_1 и S_2 равны, если
   - Совпадают их длины, $n = m$
   - $S1\ [i] = S2\ [i]$ для всех $i = 1, 2, \ldots, n$
2. Строка S_1 лексикографически меньше S_2 ($S_1 < S_2$), если выполняется одно из условий
   - $n < m$ и $S_1[1..n] = S_2[1..n]$.
   - Существует такое целое число i, $i \in [1..\min\{n, m\}]$, что
     $S_1\ [1..(i - 1)] = S_2\ [1..(i - 1)]$ и $S_1\ [i] < S_2\ [i]$,
     (i – первая позиция слева, в которой элементы строк различаются)
3. Префикс строки S, заканчивающийся в позиции i, – это подстрока $S[1..i]$
4. Суффикс строки S, начинающийся в позиции i, – это подстрока $S[i..n]$
5. Префиксы и суффиксы называются собственными, если они не являются пустыми и не совпадают с S

Пример, задача, упражнение 1

Примеры для алфавита – кодов ASCII

1. Abcd < abcd
2. abda < abdf
3. abcd < b
4. abc < abcd
5. a < bcdef
6. Abcd < b

Задача:

Даны: строка P (образец) длины *m* символов и строка T (текст) длины *n* символов, *n* >= *m*.

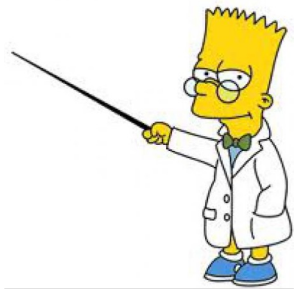
Найти все вхождения образца P в текст T.

Например, P = aab, T = aacbaabaatabaabaaw

P входит в T, начиная с позиций 5 и 13: T = aacb**aab**aataba**ab**aaw

Упражнение 1 – написать программу, определяющую

- количество вхождений подстроки P в строку T и
- номера позиций в строке T – начала вхождений.



Сложность программы, решающей «в лоб» эту задачу, O(n*m)

Цель – найти решение сложности O(n+m)

Методы предварительного анализа строк

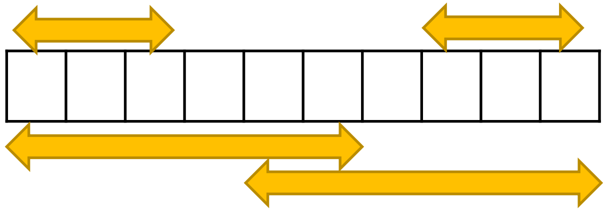
Методы предварительного анализа строк позволяют выявить их структуру – закономерности расположения символов в строке.

Важно решить задачу за линейное время.

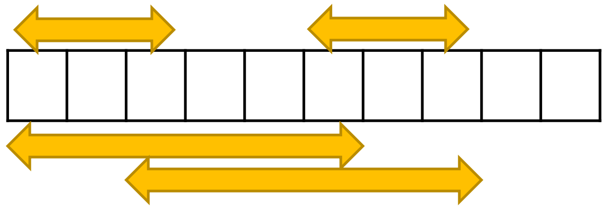
Две схемы предварительного анализа – нахождение

- граней строк,
- блоков строк.

Гранью (border, verge, brink) br строки S называется любой собственный префикс этой строки, равный суффиксу S.



Блоком (bloc) bl строки S в позиции `i (bl[i])` называется длина наибольшей подстроки S, которая начинается в `i` и совпадает с префиксом S.



Грани строки

Гранью (border, verge, brink) br строки S называется любой собственный префикс этой строки, равный суффиксу S.

Длина грани – количество символов в ней

Наибольшая грань – наибольший собственный префикс строки, равный ее суффиксу

Примеры

1. Строка S = abaababaabaab имеет две грани (не пустые) – ab и abaab.

   abaababaabaab abaababaabaab

2. Строка S = abaabaab имеет две грани (не пустые) – ab и abaab, но вторая грань – перекрывающаяся

   abaabaab abaabaab

3. Строка длины n из повторяющегося символа, например, aaaaaaaa, имеет n–1 грань. Это грани

   a, aa, aaa, aaaa, aaaaa, aaaaaa, aaaaaaa.

ЗАМЕЧАНИЕ: Понятие «собственный префикс» исключает грань, совпадающую с самой строкой.

Наибольшая грань строки. Упражнение 2

Простой алгоритм вычисления наибольшей грани строки S – последовательная проверка совпадения префиксов

$$S[1], S[1..2], S[1..3],\ldots, S[1..n-1]$$

с соответствующими суффиксами

$$S[n], S[n-1..n], S[n-2..n], \ldots, S[2..n]$$

Сложность алгоритма – $O(n^2)$

Упражнение 2. Написать программу вычисления наибольшей грани строки S.

Упражнение 3

Упражнение 3. Вычислить длины наибольших граней для всех подстрок S [1..i], i=1,..., n и сохранить их в массиве граней br[1..n] (массиве длин максимальных граней)

br[1] = 0 всегда, т.к. подстрока S[0] не имеет собственных подстрок.

Сложность алгоритма – $O(n^3)$

№	S	Массив br
1	aaaaaa	0,1,2,3,4,5
2	abcdef	0,0,0,0,0,0
3	abaababaabaab	0,0,1,1,2,3,2,3,4,5,6,4,5
4	abcabcabcabc = $(abc)^4$	0,0,0,1,2,3,4,5,6,7,8,9
5	abcabdabcabeabcabdabcabc	0,0,0,1,2,0,1,2,3,4,5,0,1,2,3,4,5,6,7,8,9,10,11, 3

Пример решения упражнения 3.5

i	S[1,i]	br[i]	i	S[1,i]	br[i]
1	a	0	13	abcabdabcabea	1
2	ab	0	14	abcabdabcabeab	2
3	abc	0	15	abcabdabcabeabc	3
4	abca	1	16	abcabdabcabeabca	4
5	abcab	2	17	abcabdabcabeabcab	5
6	abcabd	0	18	abcabdabcabeabcabd	6
7	abcabda	1	19	abcabdabcabeabcabda	7
8	abcabdab	2	20	abcabdabcabeabcabdab	8
9	abcabdabc	3	21	abcabdabcabeabcabdabc	9
10	abcabdabca	4	22	abcabdabcabeabcabdabca	10
11	abcabdabcab	5	23	abcabdabcabeabcabdabcab	11
12	abcabdabcabe	0	24	abcabdabcabeabcabdabcabc	3

Выводы из решения упражнения 3.5

Выводы из полученных результатов

- `br[i + 1] <= br[i] + 1` для любых i от 1 до (n – 1)
- Значения br возрастают с шагом не более 1
- `br[i + 1] = br[i] + 1,` когда `S[i + 1] = S[br[i] + 1]`

i	S[1,i]	br[i]
17	abcabdabcabeabcab	5
18	abcabdabcabeabcabd	6
19	abcabdabcabeabcabda	7
20	abcabdabcabeabcabdab	8
21	abcabdabcabeabcabdabc	9
22	abcabdabcabeabcabdabca	10
23	abcabdabcabeabcabdabcab	11
24	abcabdabcabeabcabdabcabc	3

Выводы из решения упражнения 3.5

Что делать, если `S[i + 1] ≠ S[br[i] + 1]` ?

Рассматривается наибольшая грань подстроки `S[1..br[i]]` и проверяется равенство `S[i+1] = S[br[br[i]] +1]`. Если неуспех, повторяем.

В упражнении 3.5

1) при `i=23` у подстроки `S[23]` длина грани = 11, грань = abcabdabcab
   Для следующей подстроки `S[23+1] ≠ S[br[i]+1]`, т.к. 'e' ≠ 'c'. Далее `i'= br[23] = 11`.

2) Проверим наибольшую грань подстроки `S[1..11]`, ее длина = 5, грань = abcab
   `S[23+1] ≠ S[br[br[i]]+1]` , т.к. 'd' ≠ 'c'. Далее `i"= br[11] = 5`.

3) Проверим наибольшую грань подстроки `S[1..5]`, ее длина = 2, грань = ab
   `S[23+1] = S[br[br[br[i]]] +1]` , т.к. 'c' = 'c'. Нашли грань для `S[24]`, длина 3, грань abc

i	S[1,i]	br[i]	i	S[1,i]	br[i]
5	abcab	2	17	abcabdabcabeabcab	5
11	abcabdabcab	5	23	abcabdabcabeabcabdabcab	11
12	abcabdabcabe	0	24	abcabdabcabeabcabdabcabc	3

Еще раз, на примере упражнения 3.3

Что делать, если `S[i+1] ≠ S[br[i] +1]` ?

Рассматривается наибольшая грань подстроки `S[1..br[i]]` и проверяется равенство

`S[i+1] = S[br[br[i]] +1]`

Рассмотрим пример 3 из таблицы

i	1	2	3	4	5	6	7	8	9	10	11	12	13
S	a	b	a	a	b	a	b	a	a	b	a	a	b
br[i]	0	0	1	1	2	3	2	3	4	5	6	4	5

```
S[12] ≠ S[br[11] + 1];
S[12] ≠ S[7]  ('a'≠'b')

S[12] = S[br[br[11]] + 1];
S[12] = S[br[6] + 1] = S[4]; 'a'='a'

br[12] = br[6] + 1 = 4
```

Задания

Задание 1. Написать программу вычисления массива граней подстрок (массива длин максимальных граней) сложности O(n)

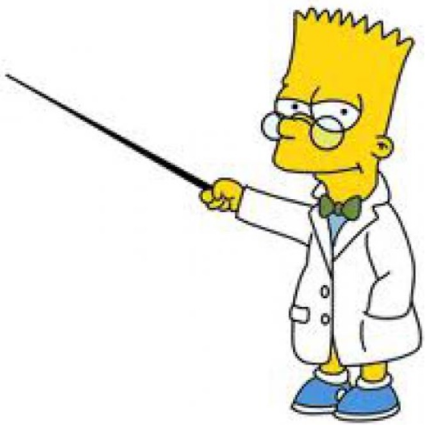
Задание 2. Найти все вхождения подстроки P в строку T (все позиции в строке T, с которых начинается вхождение подстроки P и количество вхождений) с использованием массива граней

Подсказка. Пусть имеется символ $, не входящий в алфавит A.

Сформируем из P и T строку S=P$T и вычислим массив граней максимальной длины br для всех подстрок S [1..i].

В полученном массиве br надо найти значения, равные длине P и их количество.

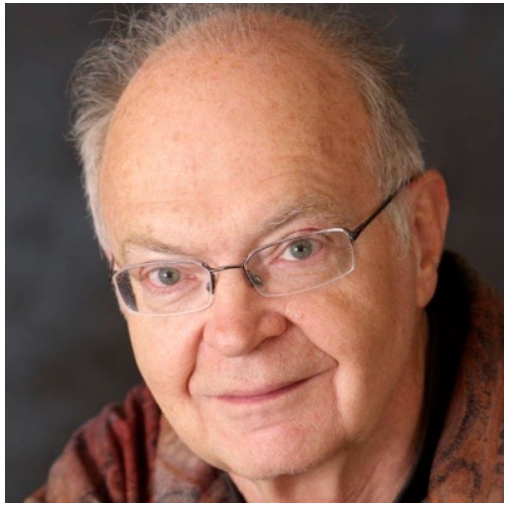
Номер позиции i в массиве br указывает на правый конец вхождения P в T.



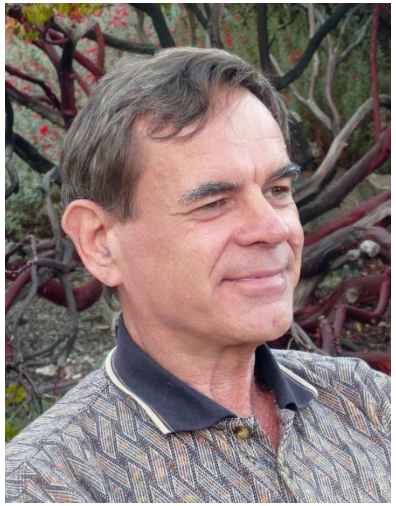
123456789012 aba$abaabaab 001012312312	1234567890123456789 ba$abbabaabbaababba 0000112120112012112

Об авторах алгоритма КМП

Алгоритм Кнута-Морриса-Пратта – один из самых известных алгоритмов поиска подстроки в строке, имеющий временную сложность O(n)



Дональд Кнут, род. 10.01.1938



Vaughan Ronald Pratt, род. 1944

James H. Morris, род. 1941

Основная идея

Дан текст T и образец P (строка T и подстрока P).

Подстрока последовательно «прикладывается» к строке и проводится пошаговое сравнение символов.

В простом алгоритме после несовпадения какой-то позиции осуществляется сдвиг на одну позицию по T.

В алгоритме К-М-П сдвиг в некоторых случаях выполняется более чем на одну позицию ЗА СЧЕТ ПРЕДВАРИТЕЛЬНОГО АНАЛИЗА ПОДСТРОКИ P (НЕ СТРОКИ T!!!)

ПРЕДВАРИТЕЛЬНО ВЫЧИСЛЯЕТСЯ МАССИВ ГРАНЕЙ ПОДСТРОКИ P

k	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
P	a	b	c	a	e	a	b	c	a	b	c	a
br	0	0	0	1	0	1	2	3	4	2	3	4

Рассмотрим на примерах. Пример1

Ищем подстроку P в строке T. Массив граней подстроки P вычислен.

`i` – номер символа в строке T, k – номер символа в подстроке P, `br[1..12]` – массив граней подстроки P.

Произошло несовпадение символов при k=9, i=11

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*
T	c	a	a	b	c	a	e	a	b	c	b	b	c	a	b	c	a	e	a	b
k			*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*						
P			a	b	c	a	e	a	b	c	a	b	c	a						
br			0	0	0	1	0	1	2	3	4	2	3	4						
P								a	b	c	a	e	a	b	c	a	b	c	a	

`br[8]=3` => суффикс длины 3 подстроки P совпадает с префиксом длины 3 подстроки P

Чтобы префикс совместить с суффиксом надо сдвинуть подстроку P на (8-3)=5 позиций. При этом гарантируется совпадение `br[8]=3` символов P с соответствующими символами T. Следующее сравнение выполняется между символами `T[11]` и `P[4]`, то есть между `T[i]` и `P[br[k-1]+1]`

`T[11] и P[br[9-1]+1] = P[br[8]+1] = P[3+1] = P[4]`

Пример 2

Заданы T и P = abcxabcde, вычислен массив граней P

```
i  =  1 2 3 4 5 6 7 8 9
P  =  a b c x a b c d e
br= (0,0,0,0,1,2,3,0,0)
```

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*
T	a	b	a	b	c	x	a	b	d	a	b	c	x	a	b	c	x	a	b	c	d	e
P	a	b	c	x	a	b	c	d	e													
P			a	b	c	x	a	b	c	d	e											
							a	b	c	x	a	b	c	d	e							
P									a	b	c	x	a	b	c	d	e					
P										a	b	c	x	a	b	c	d	e				
P														a	b	c	x	a	b	c	d	e

br[3-1] = 0
Сдвиг 2-0=2

br[7-1] = 2
Сдвиг 6-2 = 4

br[3-1] = 0
Сдвиг 2-0 = 2

Сдвиг 1

br[8-1] = 3
Сдвиг 7-3 = 4

Если k – номер последнего совпавшего символа в подстроке P, то сдвиг равен

$$\Delta = \boldsymbol{k} - \boldsymbol{br}[\boldsymbol{k}]$$

Решим пример

Псевдокод

T – строка длины n
P – подстрока длины m (образец, который ищем в T)
br[1 .. m] - массив граней подстроки P
k - индекс сравниваемой позиции в подстроке P

```
k = 0;
for i = 1 to n do  // индекс символа в T
    while (k > 0) and (P[k + 1] ≠ T[i]) do
       k = br[k];
    if (P[k + 1] = T[i])  then k = k + 1;
    if k = m then
       вывод "P входит в T с позиции", i – m + 1;
       k = br[m];
```

Что еще можно улучшить

В примере сравнивались T[9] с P[7], а затем T[9] с P[3], хотя заранее ясно, что они не совпадут

i	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2
T	a	b	a	b	c	x	a	b	d	a	b	c	x	a	b	c	x	a	b	c	d	e
P			a	b	c	x	a	b	c	d	e											
P							a	b	c	x	a	b	c	d	e							

Для того чтобы убрать лишние проверки (и «сдвиги») введем понятие уточненных граней brs[i].

Для каждой позиции i строки S **уточненной гранью brs[i]** является такой наибольший собственный суффикс S[1..i], совпадающий с префиксом S, что

$$S[i + 1] \neq S[brs[i] + 1]$$

Другими словами, следующий символ за префиксом, равным суффиксу, не должен совпадать с символом S[i + 1].

brs[n] получаем при условии, что к S приписывается символ, которого нет в алфавите.

Примеры

Для каждой позиции i строки S **уточненной гранью brs[i]** является такой наибольший собственный суффикс S[1..i], совпадающий с префиксом S, что

$$S[i+1] \neq S[brs[i]+1]$$

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
S	a	b	c	x	a	b	c	d	e
br	0	0	0	0	1	2	3	0	0
brs	0	0	0	0	0	0	3	0	0

i	S[1,i]	br[i]	S[i + 1]	S[br[i] + 1]	brs[i]
4	abcx	0			0
5	a**b**cxa **b**	1	**b**	**b**	0
6	ab**c**xab **c**	2	**c**	**c**	0
7	abc**x**abc **d**	3	**x**	**d**	3
8	abcxabcd e	0			0
9	abcxabcde [	0			0

Вычисление brs

Для вычисления используем массив граней, полученный за линейное время.

Вычисление brs проводится также за линейное время

```
brs[1] = 0;
for i = 2 to n do
    if S[br[i] + 1] ≠ S[i + 1] then
        brs[i] = br[i]
    else
        brs[i] = brs[br[i]];
```

Примеры

Для каждой позиции i строки S **уточненной гранью brs[i]** является такой наибольший собственный суффикс S[1..i], совпадающий с префиксом S, что

$$S[i + 1] \neq S[brs[i] + 1]$$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
S	a	b	a	a	b	a	b	a	a	b	a	a	b	a	b	a	a	b	a	b	a
br	0	0	1	1	2	3	2	3	4	5	6	4	5	6	7	8	9	10	11	7	8
brs	0	0	1	0	0	3	0	1	0	0	6	0	0	3	0	1	0	0	11	0	8

i	S[1,i]	br[i]	S[i + 1]	S[br[i] + 1]	brs[i]
10	abaababaab a				
11	abaababaaba a				
12	abaababaabaa b				
13	abaababaabaab a				
14	abaababaabaaba b				
15	abaababaabaabab a				

Заполните таблицу

```
00 brs[1] = 0;
01 for i = 2 to n do
02     if S[br[i] + 1] ≠ S[i + 1] then
03         brs[i] = br[i]
04     else
05         brs[i] = brs[br[i]];
```

Упражнение

Выполнить трассировку алгоритма Кнута-Морриса-Пратта для заданных T и P

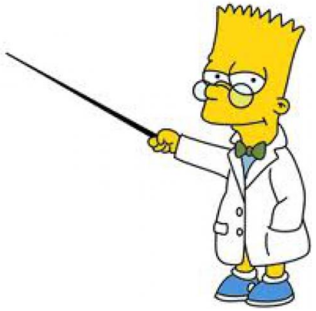
Для этого надо построить br[i] для строки P, затем brs[i], сдвиг по T определяется как $\boldsymbol{k} - \boldsymbol{brs}[\boldsymbol{k}]$, где k − номер последнего совпавшего символа в подстроке P

1) T = abbabaabbaababba P = abbab

2) T = abcabdabcabeabcabdabcabc P = abda

3) T = abcabdabcabcabcabd P = abcabc

4) T = abcabcabdabcabcabcb P = abcabcabc



Задания

1. Написать код вычисления массива длин уточненных граней за линейное время по заданному массиву длин граней

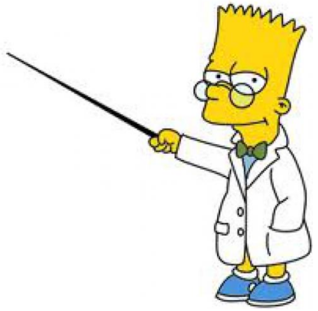
ВАЖНО!!! (Массив граней = массив длин граней)

2. Написать код алгоритма Кнута – Мориса – Пратта по массиву длин уточненных граней

В контесте: реализовать алгоритм Кнута-Морриса-Пратта с использованием brs подстроки P

Вход – строки T[n] и P[m], m<n

Выход – номера позиций вхождения P в T и количество вхождений



Псевдокод алгоритма КМП поиска вхождений подстроки в строку

```
функция Посчитать-Функцию-Префиксов(P)
m = |P|
π[1] = 0
k = 0
для i = 2 до m
    пока k > 0 и P[k + 1] ≠ P[i]
        k = π[k]
    если P[k + 1] = P[i]
        то k = k + 1
    π[i] = k
вернуть π
```

```
функция Кнут-Моррис-Пратт(T,P)
n = |T|
m = |P|
π = Посчитать-Функцию-Префиксов(P)
kol = 0 // число совпавших символов
для i = 1 до n
    пока kol > 0 и P[kol + 1] ≠ T[i]
        kol = π[kol]  // след. символ не совпадает
    если P[kol + 1] = T[i]
        то kol = kol + 1 // след. символ совпадает
    если kol = m
        то образец обнаружен – сдвиг на i-m
            kol = π[kol]
```

Сходство строк

Можно сравнить две строки на точное равенство, если в них одинаковое число элементов. Как?

А что, если надо найти в наборе похожие строки?

Какие строки считать похожими?

Количественно оценить сходство можно с помощью т. н. **расстояния редактирования** (edit distance)

Расстояние редактирования (edit distance) - минимальное количество операций

- вставки одного символа,
- удаления одного символа,
- замены одного символа на другой,

необходимых для превращения одной строки в другую.

Впервые определено В. И. Левенштейном.

Сходство строк. Применение

- обработка текстов (анализ, проверка правописания),
- автоподсказки,
- биоинформатика (sequence alignment),
- автоисправление ошибок,
- diff
- и другие

Метрики сходства строк и алгоритмы

- Расстояние Левенштейна [1965]
- Расстояние Хэмминга (Hamming) [1950]
- Расстояние Дамерау – Левенштейна (Damerau–Levenshtein) [1964]
- Сходство Джаро-Винклера (Jaro–Winkler) [1990]
- Расстояние Ли (Lee) [1958]
- Расстояние Йенсена – Шеннона (на основе метрики сходства распределений вероятности Jensen–Shannon divergence) [2003]
- Алгоритм Беза-Йейтса и Гонне (Baeza-Yates–Gonnet) или **bitap** algorithm [1964,1977,19**91**,19**96**,1998]

Расстояние Левенштейна

Владимир Иосифович Левенштейн

20 мая 1935 – 06 сен 2018, Москва

В 1958 окончил Мех-мат МГУ

и начал работать в ИПМ им. Келдыша,

там и работал.

Метрику (0-1) предложил в 1965,

статью перевели в 1966.

Часто под расстоянием редактирования вообще понимают именно эту метрику.

Расстояние Левенштейна

Редакционное предписание – последовательность операций для получения второй строки (S2) из первой (S1).

Каждая операция имеет стоимость (cost).

Стоимость предписания – сумма стоимостей всех операций для получения второй строки (S2) из первой (S1).

```
          1 2 3 4 5 6
  S1 = Г Н О М И К
  S2 =   Д О М И
```

```
        1 2 3 4 5
  S1 = С Л   О Н
  S2 = Б А Т О Н
       R R I M M
```

```
        1 2 3 4 5 6 7 8 9
  S1 = С Л О Н
  S2 =         Б А Т О Н
       D D D D I I I I I
```

Операции:

- **замена** (R, replace),
- **вставка** (I, insert),
- **удаление** (D, delete) символа.

Совпадение обозначают M (match).

Расстояние Левенштейна

Расстояние Левенштейна – это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую

a – строка длины |a| = m;
b – строка длины |b| = n;

Расстояние Левенштейна: $d(a,b) = lev_{a,b}(|a|,|b|) = lev_{a,b}(m,n)$
Вычисление lev(m,n) начнем со сравнения последних символов a[m] и b[n].
Если a[m]=b[n], то операции не нужны, а расстояние будет определяться расстоянием между подстроками a[1..(m-1)] и b[1..(n-1)], т.е. между префиксами строк a и b (в примерах ниже D – это просто символ)

Строка a	xxxxxxxxxD	xxxxxD
Строка b	yyyyD	yyyyyyyyyyyD

lev(m,n) = lev(m-1,n-1)

Перейдем к сравнению префиксов	xxxxxxxxx	xxxxx
	yyyy	yyyyyyyyyyy

Расстояние Левенштейна. Замена символа

Если a[m]≠b[n], надо выполнить одну из операций – замену, вставку или удаление символа в первой строке a.

1) Замена символа в первой строке
2) Удаление символа из первой строки
3) Вставка символа в первую строку из второй строки

`lev(m,n) = lev(m-1,n-1) + 1`

1) Замена символа

Строка a	xxxxxxxxxE	xxxxxE
Строка b	yyyyD	yyyyyyyyyyyyD
Меняем E на D	xxxxxxxxxD	xxxxxD
(1 операция)	yyyyD	yyyyyyyyyyyyD
Перейдем к	xxxxxxxxx	xxxxx
сравнению префиксов	yyyy	yyyyyyyyyyyy

Префиксы для дальнейших вычислений: a[1..(m-1)] и b[1..(n-1)]

Расстояние Левенштейна. Удаление символа

2) Удаление символа

```
Строка a            xxxxxxxxxE          xxxxxE
Строка b            yyyyD               yyyyyyyyyyyyD

Удаляем    E        xxxxxxxxx           xxxxx
(1 операция)        yyyyD               yyyyyyyyyyyyD

Перейдем к          xxxxxxxxx           xxxxx
сравнению           yyyyD               yyyyyyyyyyyyD

Префиксы для дальнейших вычислений a[1..(m-1)] и b[1..n]
```

```
lev(m,n) = lev(m-1,n) + 1
```

Расстояние Левенштейна. Вставка символа

3) Вставка символа

Строка a	xxxxxxxxxE	xxxxxE
Строка b	yyyyD	yyyyyyyyyyyyD
Вставим D	xxxxxxxxxED	xxxxxED
	yyyyD	yyyyyyyyyyyyD
Перейдем к сравнению	xxxxxxxxxE	xxxxxE
	yyyy	yyyyyyyyyyyy

Префиксы для дальнейших вычислений a[1..m] и b[1..(n-1)]

```
lev(m,n) = lev(m,n-1) + 1
```

Вычисление расстояния Левенштейна

Для префиксов строк `a[1..i]` и `b[1..j]` расстояние Левенштейна вычисляется по формуле

$$\operatorname{lev}_{a,b}(i,j)=\begin{cases}\max(i,j) & \text{if}\min(i,j)=0,\\ \min\begin{cases}\operatorname{lev}_{a,b}(i-1,j)+1\\ \operatorname{lev}_{a,b}(i,j-1)+1\\ \operatorname{lev}_{a,b}(i-1,j-1)+1_{(a_i\neq b_j)}\end{cases} & \text{otherwise.}\end{cases}$$

$$1_{(a_i\neq b_j)}=\begin{cases}0, & a_i=b_j\\ 1, & \text{otherwise.}\end{cases}$$

Поиск расстояния (количества операций) и поиск редакционного предписания (последовательность операций)— разные задачи, вторая сложнее!

Показано, что расстояние Левенштейна нельзя найти быстрее, чем за $O(n^{2-\varepsilon})$,

где n – максимальная длина строки из двух заданных,

ε – константа.

если *exponential time hypothesis* верна.

Расстояние Левенштейна

Верно, что

1. d(a,b) ≥ ||a|-|b||
2. d(a,b) ≤ max(|a|,|b|)
3. d(a,b) = 0 т. и т. т., когда a = b
4. d(a,c) ≤ d(a,b) + d(b,c) - правило треугольника

Почему?

Можно обобщить задачу, если задать цены операций (здесь i и j - обозначение символов, не индексов строк):

w(i,j) – цена замены символа i на символ j
w(ε,j) – цена вставки символа j (ε – нет символа)
w(i,ε) – цена удаления символа i

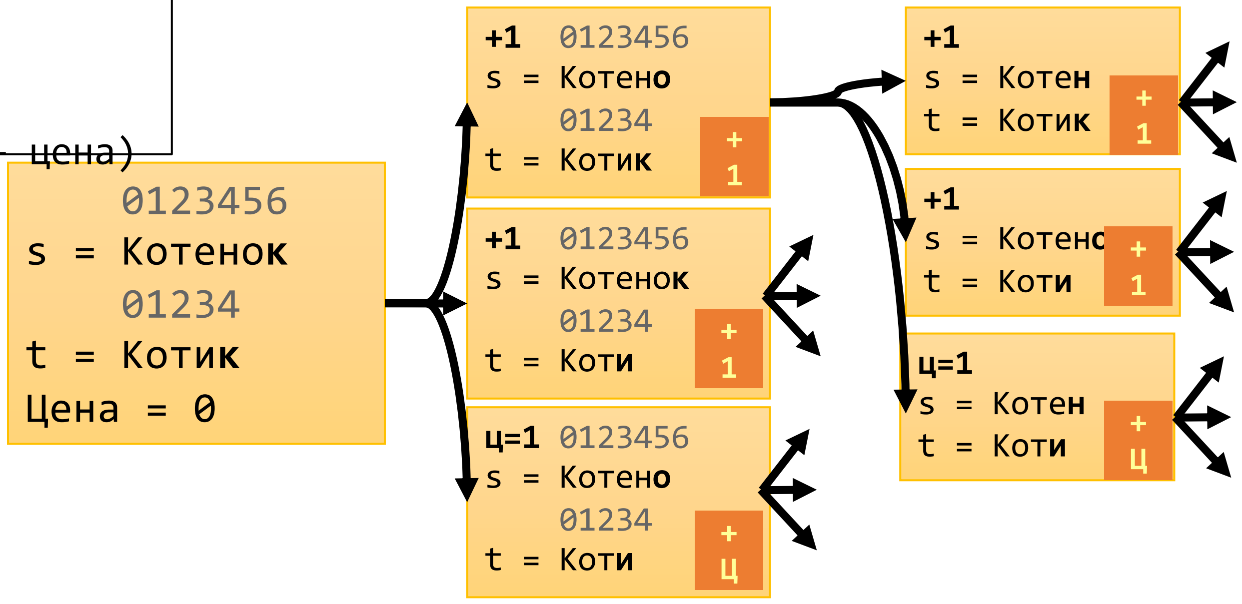
Для расстояния Левенштейна:

w(i,i) = 0,
w(i,j) = 1, если i ≠ j
w(ε,j) = 1
w(i,ε) = 1

Псевдокод и пример

```
процедура Lev(строка s, строка t)
  цена ← 0;
  если |s| = 0
     то вернуть |t|;
  если |t| = 0
     то вернуть |s|;
  если s[|s|] = t[|t|] // проверяем последний символ в строке
     то цена ← 0
     иначе цена ← 1
  вернуть минимум (Lev(s[1..|s| - 1], t) + 1,
                   Lev(s, t[1..|t| - 1]) + 1,
                   Lev(s[1..|s| - 1], t[1..|t| - 1]) + цена)
```

Примитивный рекурсивный алгоритм



Предложите свои примеры

АЛГОРИТМ ВАГНЕРА–ФИШЕРА

Алгоритм Вагнера–Фишера

Построим таблицу D(0..m,0..n)

Шаг по горизонтали – вставка символа t[j]

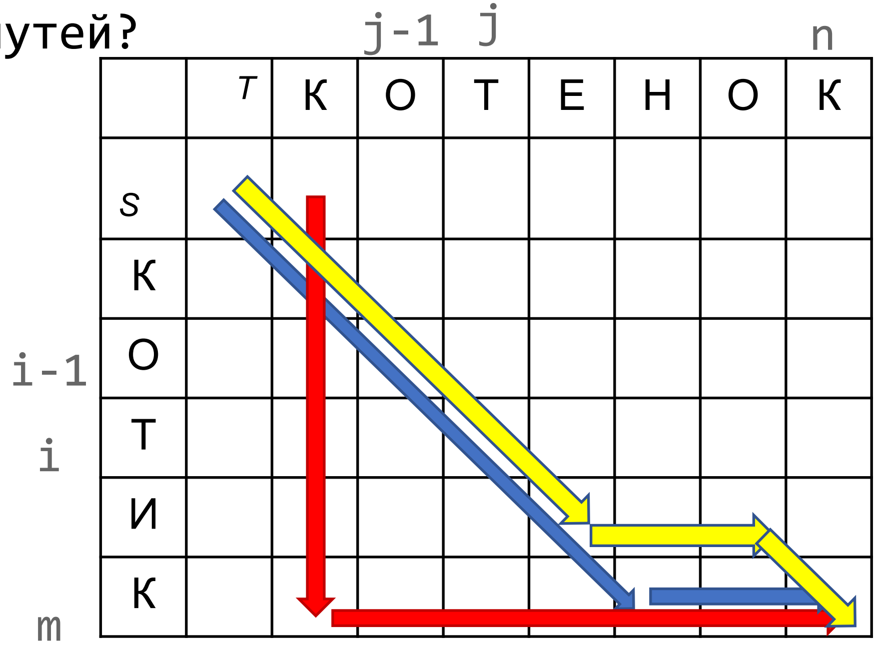
Шаг по вертикали – удаление символа s[i]

Шаг по диагонали – замена t[j] на s[i], если t[j] ≠ s[i],

иначе ничего не делаем

Путь из левого верхнего угла в правый нижний – последовательность операций.

Сколько всего путей?



Красные стрелки – 12 операций

```
КОТИК
     КОТЕНОК
DDDDDIIIIIII
```

Синие стрелки – 4 операции

```
КОТИК
КОТЕНОК
MMMRRII
```

Желтые стрелки – 3 операции

```
КОТИ  К
КОТЕНОК
MMMRIIM
```

Алгоритм Вагнера–Фишера

Для поиска расстояния – **алгоритм Вагнера-Фишера** (Wagner–Fischer, 1974)

```
процедура Lev(s[1..m],t[1..n])
d [0..m, 0..n] ← 0 // массив (m+1)x(n+1) заполняем нулями (для расстояний)
для всех i из [0..m] // i удалений из строки s, если строка t пуста
   d[i, 0] ← i
для всех j из [0..n] // j вставок из строки t, если строка s пуста
   d[0, j] ← j

для всех j из [1..n]
   для всех i из [1..m]
      если s[i] = t[j]
         то d[i, j] ← d[i-1, j-1] // все ок, символы равны
         иначе d[i, j] ← min(d[i-1, j] + 1, // 1-цена удаления символа s[i]
                             d[i, j-1] + 1, // 1-цена вставки символа t[j]
                             d[i-1, j-1]+1) // 1-цена замены символа s[i] на t[j]

вернуть d[m,n]
```

Реальнo предлагался много раз:
T.K.Винцюк (Vintsyuk) – 1968
S.B.Needleman и C.D.Wunsch - 1970
D.Sankoff – 1972
R.Wagner и M.J.Fischer - 1974
и другие

Операции Θ(mn)
Память   Θ(mn)

Память можно уменьшить до Θ(min(m,n)), если модифицировать алгоритм

Что означает число в ячейке заполняемой таблицы?

Алгоритм Вагнера–Фишера

					j-1	j			n
	d[i,j]	*T*	К	О	Т	Е	Н	О	К
	S	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	К	*1*	0	1	2	3	4	5	6
i-1	О	*2*	1	0	1	2	3	4	5
i	Т	*3*	2	1	0	1	2	3	4
	И	*4*	3	2	1	1	2	3	4
m	К	*5*	4	3	2	2	2	3	3

Строка 10
`вставка t[j]`

Строка 09
`удаление s[i]`

Строка 11
`символы равны`

Строки 07-08
`замена s[i] на t[j]`

```
07  если s[i] = t[j]
08     то d[i, j] ← d[i-1, j-1] // все ок, символы равны
09     иначе d[i, j] ← min(d[i-1, j] + 1,   // удаление s[i]
10                         d[i, j-1] + 1,   // вставка t[j]
11                         d[i-1, j-1] + 1) // замена s[i] на t[j]
```

Алгоритм Вагнера–Фишера

	T =	К	О	Т	Е	Н	О	К
S =	0	1	2	3	4	5	6	7
К	1	0	1	2	3	4	5	6
О	2	1	0	1	2	3	4	5
Т	3	2	1	0	1	2	3	4
И	4	3	2	1	1	2	3	4
К	5	4	3	2	2	2	3	3

Что означают числа в ячейках таблицы?

Значение `d[i,j]` – расстояние Левенштейна между префиксами строк `s[1..i]` и `t[1..j]`

Например,
расстояние между КО и КОТЕН `d[2,5]=3;`
расстояние между КОТИК и КОТ `d[5,3]=2;`

```
КО
КОТЕН
MMIII
```

```
КОТИК
КОТ
MMMDD
```

ВАЖНО!!! В формулах здесь используется +1, мы считаем, что цены операций равны 1.
Часто цены операций отличаются и даже могут быть отрицательными.

Алгоритм Вагнера–Фишера

D		*T*	К	О	Т	Е	Н	О	К
		0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
S	*0*	0	1	2	3	4	5	6	7
К	*1*	1	0	1	2	3			
О	*2*	2	1	0	1	2			
Т	*3*	3	2	1	0	1			
И	*4*	4	3	2	1	1			
К	*5*	5	4	3	2				

Строка 10
`вставка t[j]`

Строка 09
`удаление s[i]`

Строка 11
`символы равны`

Строки 07-08
`замена s[i] на t[j]`

```
07    если s[4] = t[4]                // не выполнено, идем на «иначе»
08      то d[4, 4] ← d[3, 3]
09      иначе d[4, 4] ← min(d[3, 4] + 1, // 1+1=2, удаление s[4] = ‘и’
10                          d[4, 3] + 1, // 1+1=2, вставка t[4] = ‘е’
11                          d[3, 3] + 1) // 0+1=1, замена s[i] на t[j] (‘и’ на ‘е’)
```

Алгоритм Вагнера–Фишера. Восстановление пути

	T =	К	О	Т	Е	Н	О	К
S =	0	1	2	3	4	5	6	7
К	1	0	1	2	3	4	5	6
О	2	1	0	1	2	3	4	5
Т	3	2	1	0	1	2	3	4
И	4	3	2	1	1	2	3	4
К	5	4	3	2	2	2	3	3

```
КОТ  ИК
КОТЕНОК
MMMIIRM
```

В таблице показан один путь.
Сколько еще вариантов путей минимальной стоимости можно найти?

Идем из правого нижнего угла `d[m,n]` в левый верхний, на каждом шаге выбираем минимальное из трех значений.

- Если минимально `d[i-1,j]+1`, добавляем удаление символа `s[i]` и идем в `(i-1,j)`.
- Если минимально `d[i,j-1]+1`, добавляем вставку символа `t[j]` и идем в `(i,j-1)`.
- Если минимально `d[i-1,j-1]+1`, добавляем замену `s[i]` на `t[j]`, если они не равны, иначе – ничего не добавляем, после чего идем в `(i-1,j-1)`.

Если минимальны два из трёх значений (или равны все три), значит, есть 2 или 3 равноценных редакционных предписания

Если одним из минимальных значений является `d[i-1,j-1]+1`, рекомендуется выбирать диагональ.

Алгоритм Вагнера–Фишера. Восстановление пути. Еще варианты

		К	О	Т	Е	Н	О	К
	0	1	2	3	4	5	6	7
К	1	0	1	2	3	4	5	6
О	2	1	0	1	2	3	4	5
Т	3	2	1	0	1	2	3	4
И	4	3	2	1	1	2	3	4
К	5	4	3	2	2	2	3	3

```
КО Т ИК
КОТЕНОК
MMIRIRM
```

		К	О	Т	Е	Н	О	К
	0	1	2	3	4	5	6	7
К	1	0	1	2	3	4	5	6
О	2	1	0	1	2	3	4	5
Т	3	2	1	0	1	2	3	4
И	4	3	2	1	1	2	3	4
К	5	4	3	2	2	2	3	3

```
КОТИК
КОТЕНОК
MMMRRII
```

		К	О	Т	Е	Н	О	К
	0	1	2	3	4	5	6	7
К	1	0	1	2	3	4	5	6
О	2	1	0	1	2	3	4	5
Т	3	2	1	0	1	2	3	4
И	4	3	2	1	1	2	3	4
К	5	4	3	2	2	2	3	3

```
КОТ ИК
КОТЕНОК
MMMIRRI
```

Не все пути оказались из 3 шагов.
Если есть два или три минимума, один из которых – диагональ, надо выбирать диагональный шаг (замена или совпадение).

Расстояние Левенштейна

Вопрос: Каковы будут значения метрики? Постройте таблицы

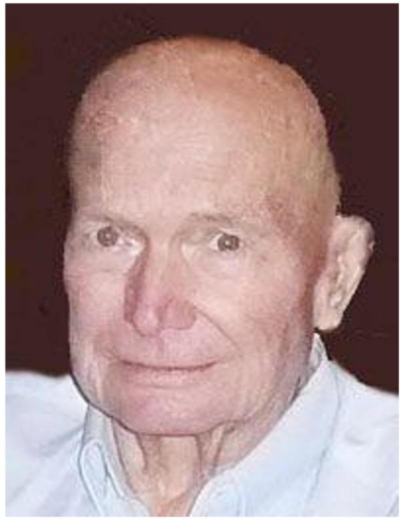
```
lev(Котенок,Котелок) = ?
lev(Котенок,Котик)   = ?
lev(Котенок,Кошка)   = ?
lev(Котенок,Кротенок)= ?
lev(Котенок,Кот)     = ?
lev(Котенок,Крот)    = ?
lev(пуля,полет)      = ?
lev(карл,коралл)     = ?
```

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j)=0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j)+1 \\ \operatorname{lev}_{a,b}(i,j-1)+1 \\ \operatorname{lev}_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

$$1_{(a_i \neq b_j)} = \begin{cases} 0, & a_i = b_j \\ 1, & \text{otherwise.} \end{cases}$$

РАССТОЯНИЕ ДАМЕРАУ–ЛЕВЕНШТЕЙНА

Расстояние Дамерау–Левенштейна



Frederick J. Damerau
25.12.1931 – 27.01.2009
IBM

Владимир Иосифович Левенштейн
20.05.1935 – 06.09.2018
РАН

Расстояние Дамерау–Левенштейна

Операции: замена (R), вставка (I), удаление (D) символа

+ **транспозиция** (T) – перестановка двух соседних символов

Совпадение обозначают M (match)

Применение: поиск опчеаток, ошиибок, отслеживание фишинговых сайтов (yandeex) и компаний adibas, анализ ДНК

a – строка длины |a|; b – строка длины |b|; $d(a,b) = d_{a,b}(|a|,|b|)$

$$d_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \\ d_{a,b}(i-2,j-2)+1 \end{cases} & \text{if } i,j > 1 \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_j \\ \min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Дамерау показал, что такие операции покрывают примерно 80-85% человеческих ошибок при печатании.

Расстояние Дамерау–Левенштейна

```
процедура DL-distance
вход: строки a[1..|a|], b[1..|b|], алфавит Σ
выход: расстояние
da ← массив [1..|Σ|] заполненный нулями
d ← массив [−1..|a|, −1..|b|] // индексирован с −1!
максрасст ← |a| + |b|
d[−1, −1] ← максрасст
для i от 0 до |a| включительно выполнить
   d[i, −1] ← максрасст
   d[i, 0] ← i
для j от 0 до |b| включительно выполнить
   d[−1, j] ← максрасст
   d[0, j] ← j
для i от 1 до |a| включительно выполнить
   db ← 0
   для j от 1 до |b| включительно выполнить
      k ← da[b[j]]
      ℓ ← db
      если a[i] = b[j]
         то цена ← 0
            db ← j
         иначе цена ← 1
      d[i, j] ← минимум(d[i−1, j−1] + цена_замены,               //замена
                        d[i,   j−1] + цена_вставки,               //вставка
                        d[i−1, j  ] + цена_удаления,              //удаление
                        d[k−1, ℓ−1] + (i-k−1) + ц_T + (j-ℓ−1)) //транспозиция
   da[a[i]] ← i
вернуть d[|a|, |b|]
```

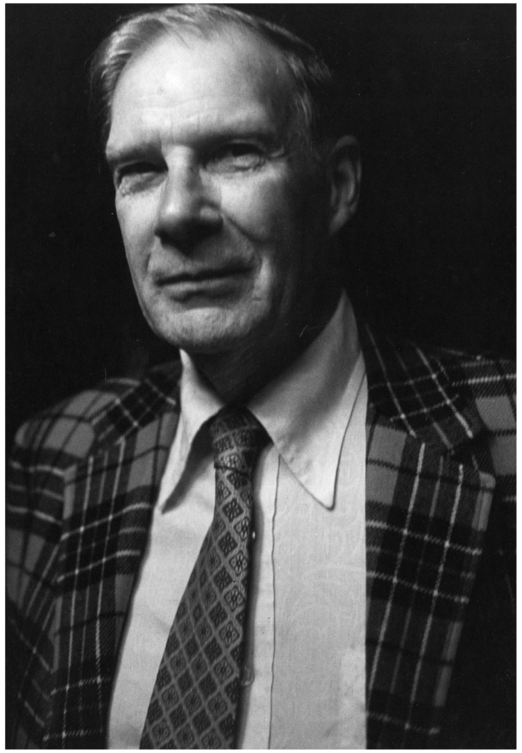
Построим **таблицу** для примеров:

КОТИЩЕ
УРОЧИЩЕ
ЧИЩЕ
КОТОФЕЙ

ДРУГИЕ МЕТРИКИ СХОДСТВА СТРОК

Расстояние Хэмминга
Расстояние Ли
Расстояние Джаро–Винклера

Расстояние Хэмминга



Richard Wesley Hamming
February 11, 1915 – January 7, 1998
Проект Манхэттен
Bell Labs (с Шенноном) с 1946 по 1976

Расстояние Хэмминга

Операции: только замена (несовпадение) (R),

Совпадение обозначают M (match).

Сравниваем две строки (векторы) **одинаковой** длины

Применение: помехозащищенное кодирование и много где еще :)

x_i x_j – две строки длины **p**, k – номер символа

Частный случай обобщенной меры расстояний Германа Минковского

(интересное: ковер Серпинского и кривая Минковского)

$$d_{ij} = \sum_{k=1}^{p} |x_{ik} - x_{jk}|.$$

		1	2	3	4	5
x_i	=	П	Ё	С	И	К
x_j	=	К	О	Т	И	К

$d_{ij} = 3$

```
процедура ham(x, y) //x,y – беззнаковые целые
расст ← 0
val ← x ^ y  // побитовое искл ИЛИ
пока val ≠ 0 // считаем установленные биты
   расст++   // нашли установленный бит
   val ← val & val – 1 // убрали его
вернуть расст // количество отличных битов
```

Расстояние Ли (C.Y. Lee)

Сравниваем две строки (векторы) – *x* и *y* одинаковой длины *n*

Алфавит {0, 1, …, q-1}, q ≥ 2 (q – количество символов)

Применение: кодирование

$$\sum_{i=1}^{n} \min(|x_i - y_i|, q - |x_i - y_i|)$$

```
    1 2 3 4 5
x = 1 0 1 1 0
y = 0 0 1 1 1
```

ld_{ij} =

```
min(|1-0|,2-|1-0|) +
min(|0-0|,2-|0-0|) +
min(|1-1|,2-|1-1|) +
min(|1-1|,2-|1-1|) +
min(|0-1|,2-|0-1|) = 1 + 0 + 0 + 0 + 1 = 2
```

При q=2 и q=3 совпадает с расстоянием Хэмминга

Расстояние Джаро–Винклера

Matthew A. Jaro
MatchWare Technologies, Inc.

William E. Winkler
US Census Bureau

Опубликовано в 1990-м году в работе W.E.Winkler – «*String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage*»

Расстояние Джаро–Винклера

Сравниваем две строки s_i и s_j разной длины
Применение: поиск дубликатов (одинаковых записей)
Два символа совпадают, если они равны и находятся друг от друга не дальше, чем

$$\left\lfloor \frac{\max(|s_1|, |s_2|)}{2} \right\rfloor - 1$$

Метрика Джаро (Jaro) — d_j

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

Метрика Jaro-Winkler — d_w

$$d_w = d_j + (\ell p(1 - d_j))$$

m — количество совпавших символов
t — половина числа перестановок
l — длина общего префикса от старта строки до максимума в 4 символа
p — константа, обозначает значительность общего префикса в общей оценке (обычно берут 0,1)

Расстояние Джаро–Винклера

S_1 = MARTHA
S_2 = MARHTA

m = 6
$|S_1|$ = 6
$|S_2|$ = 6
Макс расст (6/2)-1 = **2**

	M	A	R	T	H	A
M	1	0	0	0	0	0
A	0	1	0	0	0	0
R	0	0	1	0	0	0
H	0	0	0	0	1	0
T	0	0	0	1	0	0
A	0	0	0	0	0	1

Два символа совпадают, если они равны и не дальше, чем $\left\lfloor \frac{\max(|s_1|,|s_2|)}{2} \right\rfloor - 1$

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

d_j = 1/3 * (6/6 + 6/6 + (6-1)/6) = 0,9(4)

$$d_w = d_j + (\ell p(1 - d_j))$$

l = 3
d_w = 0,9(4) + (3 * 0,1 (1 – 0,9(4))) = 0,96

Магическое число
p = 0,1