Лекция 10 от 11.02.2016

Расстояние редактирования (расстояние Левенштейна)

Пусть у нас есть два слова и мы хотим из первого сделать второе. При этом мы можем произвольно расставлять пробелы и заменять буквы. Например:

$$
\begin{array}{llll}
\text{п е р в о е} & \cancel{\text{п}}\,\cancel{\text{е}}\,\cancel{\text{р}}\,\cancel{\text{в}}\,\cancel{\text{о}}\,\cancel{\text{е}} & & \text{п е р }\cancel{\text{в}}\text{ о е} \\
\downarrow\downarrow\downarrow\downarrow & & \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow & \downarrow\downarrow\downarrow \\
\text{в т о р о е} & & \text{в т о р о е} & \text{в т о р}\ \ \text{о е}
\end{array}
$$

Применяется это почти повсеместно:

- Проверка орфографии. Введённое слово сравнивается с теми, которые есть в словаре. Если в слове есть ошибка, то программа предложит заменить слово на наиболее похожее верное слово.

- Проверка работ на плагиат. Опять же, идёт сравнение фраз с фразами из ранее написанных работ.

- Также его используют для изучения и сравнения последовательностей ДНК.

Теперь формализуем ранее сказанное. Пусть даны строки $s = s_1 \ldots s_m$ и $t = t_1 \ldots t_n$. Тогда *выравнивание* M — множество пар (i,j) таких, что $1 \leqslant i \leqslant m, 1 \leqslant i \leqslant m$, при этом:

$$(i_1, j_1) \in M, (i_2, j_2) \in M, i_1 = i_2 \implies j_1 = j_2$$

$$(i_1, j_1) \in M, (i_2, j_2) \in M, i_1 < i_2 \implies j_1 < j_2$$

Расстояние редактирования (оно же расстояние Левенштейна) — минимальное число операций, переводящее s в t. Доступные операции — удаление букв, добавление букв и замена одной буквы на другую.

Сколько вообще существует возможных выравниваний?

Для простоты положим, что $n = m$.

- Так как каждую букву слова можно либо заменить, либо убрать, то их явно не меньше, чем 2^n.

- Пусть мы выбрали k букв первого слова, которые будут сопоставлены k буквам другого слова. Тогда получаем, что есть $(C_n^k)^2$ выравниваний для фиксированного k (так как мы вольны в выборе этих букв). Следовательно, всего существует $\sum\limits_{k=0}^{n} (C_n^k)^2$ выравниваний.

- Или по-другому: есть $2n$ букв. Из них надо выбрать k букв первого слова для замены и $n - k$ букв второго слова для вставки. Тогда существует C_{2n}^n выравниваний.

Пусть $d(s, t)$ — расстояние редактирования между s и t.

Рассмотрим первые буквы слов. У нас есть три варианта

1. Удалить s_1 и как-то превратить остаток первого слова во второе;

2. Удалить t_1 и превратить первое слово в остаток второго;

3. Сопоставить s_1 и t_1 и превратить то, что осталось, друг в друга.

Логично, что нужно выбрать наиболее оптимальный вариант. Тогда значение расстояния будет равно минимальному из этих трёх.

Базовый случай — когда одно из слов пустое. Тогда достаточно вставить все буквы из непустого слова. Следовательно, $d(w, \text{``''}) = d(\text{``''}, w) = |w|$

В итоге получаем, что расстояние Левенштейна определяется следующим образом:

$$d(s_1 \dots s_m, t_1 \dots t_n) = \begin{cases} |t|, & s = \text{`` ''} \\ |s|, & t = \text{`` ''} \\ \min \begin{cases} d(s_1 \dots s_{m-1}, t_1 \dots t_n) + 1 \\ d(s_1 \dots s_m, t_1 \dots t_{n-1}) + 1 \\ d(s_1 \dots s_{m-1}, t_1 \dots t_{n-1}) + (s_1 = t_1) \end{cases} & \text{иначе} \end{cases}$$

Составим таблицу размером $(m+1) \times (n+1)$, где $T[i][j] = d(s_i \dots s_m, t_j \dots t_n)$. Пользуясь базовыми случаями заполним верхнюю строку и правый столбец. Теперь заполним $T[6][6]$, как минимум из $1 + T[6][7], 1 + T[7][6]$ и $T[7][7] + (s_6 \neq t_6)$. Будем продолжать так до тех пор, пока не дойдём до ячейки $T[1][1]$, в которой и будет записано искомое расстояние Левенштейна:

“”	6	5	4	3	2	1	0
е	5	4	3	2	1	0	1
о	4	3	2	1	0	1	2
в	3	3	2	1	1	2	3
р	4	3	2	2	2	3	4
е	4	3	3	3	3	4	5
п	**4**	4	4	4	4	5	6
	в	т	о	р	о	е	“”

Из данной таблицы видно, что $d(\text{“первое”}, \text{“второе”}) = 4$.

Напишем псевдокод для этого алгоритма:

Algorithm 1 Нахождение расстояния Левенштейна

function EditDistance(s, t) ▷ p и q — слова
  create $T[1..(m+1), 1..(n+1)]$
  for $k := 1$ **to** $m+1$ **do**
    $T[k][n+1] = m + 1 - k$
  for $k := 1$ **to** $n+1$ **do**
    $T[m+1][k] = n + 1 - k$
  for $j := n$ **downto** 1 **do**
    for $i := m$ **downto** 1 **do**
      $T[i][j] := \min(1 + T[i+1][j], 1 + T[i][j+1], T[i+1][j+1] + (s[i] \neq t[i]))$
  return $T[1][1]$

Данный алгоритм можно улучшить, если рассматривать отдельно случаи, когда две соседние буквы переставлены местами. Тогда

$$d(s_1 \dots s_m, t_1 \dots t_n) = \begin{cases} |t|, & s = \text{“ ”} \\ |s|, & t = \text{“ ”} \\ \min \begin{cases} d(s_1 \dots s_{m-1}, t_1 \dots t_n) + 1 \\ d(s_1 \dots s_m, t_1 \dots t_{n-1}) + 1 \\ d(s_1 \dots s_{m-1}, t_1 \dots t_{n-1}) + (s_1 = t_1) \\ d(s_1 \dots s_{m-2}, t_1 \dots t_{n-2}) + 1 \end{cases} & \text{если } s_i = t_{i-1} \text{ и } s_{i-1} = t_i \\ \min \begin{cases} d(s_1 \dots s_{m-1}, t_1 \dots t_n) + 1 \\ d(s_1 \dots s_m, t_1 \dots t_{n-1}) + 1 \\ d(s_1 \dots s_{m-1}, t_1 \dots t_{n-1}) + (s_1 = t_1) \end{cases} & \text{иначе} \end{cases}$$

Данная функция называется *расстоянием Дамерау-Левенштейна.*

Сравнение алгоритмов

	Интервалы	Ширина	Редактирование
Число подзадач	$O(n)$	$O(n)$	$O(n^2)$
Число подзадач, от которых зависит задача	2	$O(n)$	3
Время	$O(n)$	$O(n^2)$	$O(n^2)$