Расстояние Левенштейна

Материал из Википедии — свободной энциклопедии

Текущая версия страницы пока не проверялась опытными участниками и может значительно отличаться от версии, проверенной 6 февраля 2016; проверки требует 1 правка.

Расстояние Левенштейна (также **редакционное расстояние** или **дистанция редактирования**) между двумя строками в теории информации и компьютерной лингвистике — это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Впервые задачу упомянул в 1965 году советский математик Владимир Иосифович Левенштейн при изучении последовательностей $0-1$.[1] Впоследствии более общую задачу для произвольного алфавита связали с его именем. Большой вклад в изучение вопроса внёс Дэн Гасфилд.[2]

Содержание

Применение

Расстояние Левенштейна и его обобщения активно применяется:

- для исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи).
- для сравнения текстовых файлов утилитой diff и ей подобными. Здесь роль «символов» играют строки, а роль «строк» — файлы.
- в биоинформатике для сравнения генов, хромосом и белков.

С точки зрения приложений определение расстояния между словами или текстовыми полями по Левенштейну обладает следующими недостатками:

1. При перестановке местами слов или частей слов получаются сравнительно большие расстояния;
2. Расстояния между совершенно разными короткими словами оказываются небольшими, в то время как расстояния между очень похожими длинными словами оказываются значительными.

Редакционное предписание

Редакционным предписанием называется последовательность действий, необходимых для получения из первой строки второй кратчайшим образом. Обычно действия обозначаются так: **D** (англ. *delete*) — удалить, **I** (англ. insert) — вставить, **R** (*replace*) — заменить, **M** (*match*) — совпадение.

Например, для 2-х строк «CONNECT» и «CONEHEAD» можно построить следующую таблицу преобразований:

M	**M**	**M**	**R**	**I**	**M**	**R**	**R**
C	**O**	**N**	**N**		**E**	**C**	**T**
C	O	N	**E**	**H**	E	**A**	**D**

Найти только расстояние Левенштейна — более простая задача, чем найти ещё и редакционное предписание (подробнее см. ниже).

Обобщения

Разные цены операций

Цены операций могут зависеть от вида операции (вставка, удаление, замена) и/или от участвующих в ней символов, отражая разную вероятность мутаций в биологии[3], разную вероятность разных ошибок при вводе текста и т. д. В общем случае:

- w(a, b) — цена замены символа a на символ b
- w(ε, b) — цена вставки символа b
- w(a, ε) — цена удаления символа a

Необходимо найти последовательность замен, минимизирующую суммарную цену. Расстояние Левенштейна является частным случаем этой задачи при

- w(a, a) = 0
- w(a, b) = 1 при a≠b
- w(ε, b) = 1
- w(a, ε) = 1

Как частный случай, так и задачу для произвольных w, решает алгоритм Вагнера — Фишера, приведённый ниже. Здесь и ниже мы считаем, что все w неотрицательны, и действует правило треугольника: если две последовательные операции можно заменить одной, это не ухудшает общую цену (например, заменить символ x на y, а потом с y на z не лучше, чем сразу x на z).

Транспозиция

Если к списку разрешённых операций добавить транспозицию (два соседних символа меняются местами), получается расстояние Дамерау — Левенштейна. Для неё также существует алгоритм, требующий O(MN) операций. Дамерау показал, что 80 % ошибок при наборе текста человеком являются транспозициями. Кроме того, расстояние Дамерау — Левенштейна используется и в биоинформатике.

Формула

Здесь и далее считается, что элементы строк нумеруются с первого, как принято в математике, а не с нулевого, как принято в языках C, C++, C#, Java.

Пусть S_1 и S_2 — две строки (длиной M и N соответственно) над некоторым алфавитом, тогда редакционное расстояние (расстояние Левенштейна) $\mathrm{d}(S_1, S_2)$ можно подсчитать по следующей рекуррентной формуле

$\mathrm{d}(S_1, S_2) = D(M, N)$, где

$$D(i,j) = \begin{cases} 0, & i = 0,\ j = 0 \\ i, & j = 0,\ i > 0 \\ j, & i = 0,\ j > 0 \\ \min\{ & \\ \qquad D(i, j-1) + 1, & \\ \qquad D(i-1, j) + 1, & j > 0,\ i > 0 \\ \qquad D(i-1, j-1) + \mathrm{m}(S_1[i], S_2[j]) & \\ \} & \end{cases},$$

где $\mathrm{m}(a, b)$ равна нулю, если $a = b$ и единице в противном случае; $\min\{\, a, b, c\,\}$ возвращает наименьший из аргументов.

Здесь шаг по i символизирует удаление (D) из первой строки, по j — вставку (I) в первую строку, а шаг по обоим индексам символизирует замену символа (R) или отсутствие изменений (M).

Очевидно, справедливы следующие утверждения:

- $\mathrm{d}(S_1, S_2) \geqslant \big||S_1| - |S_2|\big|$
- $\mathrm{d}(S_1, S_2) \leqslant \max\big(|S_1|, |S_2|\big)$
- $\mathrm{d}(S_1, S_2) = 0 \Leftrightarrow S_1 = S_2$

Пример работы алгоритма.

		P	**O**	**L**	**Y**	**N**	**O**	**M**	**I**	**A**	**L**
	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	7
L	11	10	9	8	9	8	8	8	8	7	6

Доказательство

Рассмотрим формулу более подробно. Очевидно, что редакционное расстояние между двумя пустыми строками равно нулю. Так же очевидно то, что чтобы получить пустую строку из строки длиной i, нужно совершить i операций удаления, а чтобы получить строку длиной j из пустой, нужно произвести j операций вставки.

Осталось рассмотреть нетривиальный случай, когда обе строки непусты.

Для начала заметим, что в оптимальной последовательности операций, их можно произвольно менять местами. В самом деле, рассмотрим две последовательные операции:

- Две замены одного и того же символа — неоптимально (если мы заменили x на y, потом y на z, выгоднее было сразу заменить x на z).
- Две замены разных символов можно менять местами
- Два стирания или две вставки можно менять местами
- Вставка символа с его последующим стиранием — неоптимально (можно их обе отменить)
- Стирание и вставку разных символов можно менять местами
- Вставка символа с его последующей заменой — неоптимально (излишняя замена)
- Вставка символа и замена другого символа меняются местами
- Замена символа с его последующим стиранием — неоптимально (излишняя замена)
- Стирание символа и замена другого символа меняются местами

Пусть S_1 кончается на символ «a», S_2 кончается на символ «b». Есть три варианта:

1. Символ «a», на который кончается S_1, в какой-то момент был стёрт. Сделаем это стирание первой операцией. Тогда мы стёрли символ «a», после чего превратили первые $i-1$ символов S_1 в S_2 (на что потребовалось $D(i-1,\ j)$ операций), значит, всего потребовалось $D(i-1,\ j)+1$ операций
2. Символ «b», на который кончается S_2, в какой-то момент был добавлен. Сделаем это добавление последней операцией. Мы превратили S_1 в первые $j-1$ символов S_2, после чего добавили «b». Аналогично предыдущему случаю, потребовалось $D(i,\ j-1)+1$ операций.
3. Оба предыдущих утверждения неверны. Если мы добавляли символы справа от финального «a», то, чтобы сделать последним символом «b», мы должны были или в какой-то момент добавить его (но тогда утверждение 2 было бы верно), либо заменить на него один из этих добавленных символов (что тоже невозможно, потому что добавление символа с его последующей заменой неоптимально). Значит, символов справа от финального «a» мы не добавляли. Самого финального «a» мы не стирали, поскольку утверждение 1 неверно. Значит, единственный способ изменения последнего символа — его замена. Заменять его 2 или больше раз неоптимально. Значит,
   1. Если $a=b$, мы последний символ не меняли. Поскольку мы его также не стирали и не приписывали ничего справа от него, он не влиял на наши действия, и, значит, мы выполнили $D(i-1,\ j-1)$ операций.
   2. Если $a \neq b$, мы последний символ меняли один раз. Сделаем эту замену первой. В дальнейшем, аналогично предыдущему случаю, мы должны выполнить $D(i-1,\ j-1)$ операций, значит, всего потребуется $D(i-1,\ j-1)+1$ операций.

Алгоритм Вагнера — Фишера

Для нахождения кратчайшего расстояния необходимо вычислить матрицу D, используя вышеприведённую формулу. Её можно вычислять как по строкам, так и по столбцам. Псевдокод алгоритма:

```
для всех i от 0 до M
  для всех j от 0 до N
    вычислить D(i, j)
вернуть D(M,N)
```

Или в более развёрнутом виде, и при произвольных ценах замен, вставок и удалений:

```
D(0,0) = 0
для всех j от 1 до N
  D(0,j) = D(0,j-1) + цена вставки символа S2[j]
для всех i от 1 до M
  D(i,0) = D(i-1,0) + цена удаления символа S1[i]
  для всех j от 1 до N
    D(i,j) = min{
       D(i-1, j) + цена удаления символа S1[i],
       D(i, j-1) + цена вставки символа S2[j],
```

```
        D(i-1, j-1) + цена замены символа S1[i] на символ S2[j]
    }
вернуть D(M,N)
```

(Напоминаем, что элементы строк нумеруются с *первого*, а не с нулевого.)

Для восстановления редакционного предписания требуется вычислить матрицу D, после чего идти из правого нижнего угла (M,N) в левый верхний, на каждом шаге ища минимальное из трёх значений:

- если минимально (D(i-1, j) + цена удаления символа S1[i]), добавляем удаление символа S1[i] и идём в (i-1, j)
- если минимально (D(i, j-1) + цена вставки символа S2[j]), добавляем вставку символа S2[j] и идём в (i, j-1)
- если минимально (D(i-1, j-1) + цена замены символа S1[i] на символ S2[j]), добавляем замену S1[i] на S2[j] (если они не равны; иначе ничего не добавляем), после чего идём в (i-1, j-1)

Здесь (i, j) — клетка матрицы, в которой мы находимся на данном шаге. Если минимальны два из трёх значений (или равны все три), это означает, что есть 2 или 3 равноценных редакционных предписания.

Этот алгоритм называется алгоритмом Вагнера — Фишера. Он предложен Р. Вагнером (R. A. Wagner) и М. Фишером (M. J. Fischer) в 1974 году.[4]

Память

Алгоритм в виде, описанном выше, требует $\Theta(M \cdot N)$ операций и такую же память. Последнее может быть неприятным: так, для сравнения файлов длиной в 10^5 строк потребуется около 40 гигабайт памяти.

Если требуется только расстояние, легко уменьшить требуемую память до $\Theta(\min\{M, N\})$. Для этого надо учесть, что после вычисления любой строки предыдущая строка больше не нужна. Более того, после вычисления D(i, j) не нужны также D(i-1,0) … D(i-1,j-1). Поэтому алгоритм можно переписать как

```
для всех i от 0 до M
  для всех j от 0 до N
    вычислить D(i, j)
  если i>0
    стереть строку D(i-1)
вернуть D(M, N)
```

или даже

```
для всех i от 0 до M
  для всех j от 0 до N
    вычислить D(i, j)
    если i>0 и j>0
      стереть D(i-1, j-1)
вернуть D(M, N)
```

Если требуется редакционное предписание, экономия памяти усложняется.

Для того, чтобы обеспечить время $\Theta(M \cdot N)$ при памяти $\Theta(\min\{M, N\})$, определим матрицу E минимальных расстояний между *суффиксами* строк, то есть E(i, j) — расстояние между последними i символами S_1 и последними j символами S_2. Очевидно, матрицу E можно вычислить аналогично матрице D, и так же быстро.

Теперь опишем алгоритм, считая, что S_2 — кратчайшая из двух строк.

- Если длина одной из строк (или обеих) не больше 1, задача тривиальна. Если нет, выполним следующие шаги.

- Разделим строку S_1 на две подстроки длиной $M/2$. (Если M нечётно, то длины подстрок будут $(M-1)/2$ и $(M+1)/2$.) Обозначим подстроки S_1^- и S_1^+.
- Для S_1^- вычислим последнюю строку матрицы D, а для S_1^+ — последнюю строку матрицы E.
- Найдём i такое, что $D(|S_1^-|, i) + E(|S_1^+|, N-i)$ минимально. Здесь D и E — матрицы из предыдущего шага, но мы используем только их последние строки. Таким образом, мы нашли разбиение S_2 на две подстроки, минимизирующее сумму расстояния левой половины S_1 до левой части S_2 и расстояния правой половины S_1 до правой части S_2. Следовательно, левая подстрока S_2 соответствует левой половине S_1, а правая — правой.
- Рекурсивно ищем редакционное предписание, превращающее S_1^- в левую часть S_2 (то есть в подстроку $S2[1...i]$)
- Рекурсивно ищем редакционное предписание, превращающее S_1^+ в правую часть S_2 (то есть в подстроку $S2[i+1...N]$).
- Объединяем оба редакционных предписания.[5]

Время выполнения удовлетворяет (с точностью до умножения на константу) условию

$$T(M,N) = MN + T(M/2, N') + T(M/2, N-N'),\ 0 \leq N' \leq N,$$

откуда следует (доказывается индукцией по M)

$$T(M,N) \leq 2MN$$

следовательно

$$T(M,N) = \Theta(M \cdot N)$$

Требуемая память пропорциональна $N + N/2 + N/4 + \ldots = 2N$

Кроме того, есть алгоритмы, экономящие память за счёт существенного замедления, например, время становится кубическим, а не квадратичным, по длине строк.

См. также

- Алгоритм Хирчберга
- Расстояние Дамерау — Левенштейна
- Расстояние Йенцена — Шаннона
- Расстояние Хэмминга
- Фонетический поиск
- Soundex-метод

Примечания

1. ↑ В. И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965. 163.4:845-848.
2. ↑ Гасфилд. Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. Невский Диалект БВХ-Петербург, 2003.
3. ↑ См., например: http://www.medlit.ru/medrus/mg/mg080237.htm
4. ↑ R. A. Wagner, M. J. Fischer. The string-to-string correction problem. J. ACM 21 1 (1974). P. 168—173

5. ↑ При этом во втором редакционном предписании нужно увеличить номера символов первой строки на $|S_1^-|$, а второй строки на i, поскольку теперь они отсчитываются с начала строк, а не с их середины.

Ссылки

- Визуализатор алгоритма

- Нечёткий поиск в тексте и словаре
- Пошаговое объяснение алгоритма на примере и код на Python

Источник — «https://ru.wikipedia.org/w/index.php?title=Расстояние_Левенштейна&oldid=81834473»
