Расстояние Левенштейна

Александр Чернышёв

МГУ имени М. В. Ломоносова

16 ноября 2018 г.

Содержание

Что это такое?

Расстояние Левенштейна — метрика близости двух строчек.

Что это такое?

Расстояние Левенштейна — метрика близости двух строчек.

Нам доступны три операции над строкой:

1. вставка символа: abcd → abcbd
2. удаление символа: abcd → abd
3. замена символа: abcd → abed

Что это такое?

Расстояние Левенштейна — метрика близости двух строчек.

Нам доступны три операции над строкой:

1. вставка символа: abcd → abcbd
2. удаление символа: abcd → abd
3. замена символа: abcd → abed

Тогда *расстояние Левенштейна* между двумя строками — минимальное число операций, необходимое для превращения одной строки в другую.

Пример

кошка $\xrightarrow{?}$ собака

Пример

кошка $\xrightarrow{?}$ собака

кошка $\xrightarrow{\text{добавим 'а'}}$ кошака

Пример

$$\text{кошка} \xrightarrow{?} \text{собака}$$

$$\text{кошка} \xrightarrow{\text{добавим 'а'}} \text{кошака} \xrightarrow{\text{'ш' на 'б'}} \text{кобака}$$

Пример

кошка $\xrightarrow{?}$ собака

кошка $\xrightarrow{\text{добавим 'а'}}$ кошака $\xrightarrow{\text{'ш' на 'б'}}$ кобака $\xrightarrow{\text{'к' на 'с'}}$ собака

Пример

кошка $\xrightarrow{?}$ собака

кошка $\xrightarrow{\text{добавим 'а'}}$ кошака $\xrightarrow{\text{'ш' на 'б'}}$ кобака $\xrightarrow{\text{'к' на 'с'}}$ собака

За меньшее число операций нельзя $\Rightarrow$ расстояние Левенштейна между строками «кошка» и «собака» равно 3

Зачем оно нужно?

Применяется:

- для исправления ошибок в слове;
- для сравнения текстовых файлов утилитой diff;
- в биоинформатике для сравнения генов, хромосом и белков.

Зачем оно нужно?

Применяется:

- для исправления ошибок в слове;
- для сравнения текстовых файлов утилитой diff;
- в биоинформатике для сравнения генов, хромосом и белков.

Недостатки:

- перестановка букв внутри слова порождает большое расстояние;
- между короткими словами расстояние в среднем меньше.

Как считать?

Обозначим за $d(s, t)$ расстояние Левенштейна между строками s и t с длинами n и m соответственно.

Как считать?

Обозначим за $d(s,t)$ расстояние Левенштейна между строками s и t с длинами n и m соответственно. Тогда вот формула. Берите и считайте:

$d(s,t) = D(n,m)$, где

$$D(i,j) = \begin{cases} 0, & i = 0,\ j = 0 \\ i, & i > 0,\ j = 0 \\ j, & i = 0,\ j > 0 \\ \min\{ & \\ \qquad D(i, j-1) + 1, & \\ \qquad D(i-1, j) + 1, & i > 0,\ j > 0 \\ \qquad D(i-1, j-1) + \mathbb{I}[s_i \neq t_j] & \\ \} & \end{cases}$$

Краткие пояснения

Очевидно: не выгодно делать две замены одного символа подряд и т.п.

Краткие пояснения

Очевидно: не выгодно делать две замены одного символа подряд и т.п.
По индексу i будем символы удалять из s, а по j — добавлять в s.

Краткие пояснения

Очевидно: не выгодно делать две замены одного символа подряд и т.п.
По индексу i будем символы удалять из s, а по j — добавлять в s.

Рассмотрим последние символы a и b строк s и t. Есть 3 пути:

Краткие пояснения

Очевидно: не выгодно делать две замены одного символа подряд и т.п. По индексу i будем символы удалять из s, а по j — добавлять в s.

Рассмотрим последние символы a и b строк s и t. Есть 3 пути:

1. Когда-то мы удалим a. Сделаем сразу $\Rightarrow D(i-1, j) + 1$;

Краткие пояснения

Очевидно: не выгодно делать две замены одного символа подряд и т.п. По индексу i будем символы удалять из s, а по j — добавлять в s.

Рассмотрим последние символы a и b строк s и t. Есть 3 пути:

1. Когда-то мы удалим a. Сделаем сразу $\Rightarrow D(i-1,j)+1$;
2. Когда-то мы добавим b. Сделаем в конце $\Rightarrow D(i,j-1)+1$;

Краткие пояснения

Очевидно: не выгодно делать две замены одного символа подряд и т.п. По индексу i будем символы удалять из s, а по j — добавлять в s.

Рассмотрим последние символы a и b строк s и t. Есть 3 пути:

1. Когда-то мы удалим a. Сделаем сразу $\Rightarrow D(i-1, j) + 1$;
2. Когда-то мы добавим b. Сделаем в конце $\Rightarrow D(i, j-1) + 1$;
3. Не будем делать ни то и ни другое. Тогда заменим сразу a на b (если нужно) $\Rightarrow D(i-1, j-1) + \mathbb{I}[s_i \neq t_j]$

Ок, формула есть, но как по ней считать ответ?

Ответ — алгоритм Вагнера–Фишера.

Ок, формула есть, но как по ней считать ответ?

Ответ — алгоритм Вагнера–Фишера.

Насчитываем $D(i, j)$ сначала по i от 1 до n и вложенно по j от 1 до m. Значения $D(i, j)$ при $i = 0$ или $j = 0$ заполняем в самом начале.

Ок, формула есть, но как по ней считать ответ?

Ответ — алгоритм Вагнера–Фишера.

Насчитываем $D(i, j)$ сначала по i от 1 до n и вложенно по j от 1 до m. Значения $D(i, j)$ при $i = 0$ или $j = 0$ заполняем в самом начале.

Алгоритм требует $O(nm)$ операций и такую же память. Потребление памяти можно легко сократить до $O(\min\{n, m\})$, если хранить только последнюю строку (или столбец) от насчитанной матрицы D.

А как восстановить цепочку превращений строк?

Такая цепочка превращений называется *редакционным предписанием*.

А как восстановить цепочку превращений строк?

Такая цепочка превращений называется *редакционным предписанием*.

В простейшем случае стартуем из клетки (n, m) матрицы D. Переходим в одну из клеток $(n-1, m)$, $(n, m-1)$ или $(n-1, m-1)$ в зависимости от того, куда мы переходим в формуле. И так далее.

А как восстановить цепочку превращений строк?

Такая цепочка превращений называется *редакционным предписанием*.

В простейшем случае стартуем из клетки (n, m) матрицы D. Переходим в одну из клеток $(n-1, m)$, $(n, m-1)$ или $(n-1, m-1)$ в зависимости от того, куда мы переходим в формуле. И так далее.

Но в таком случае нам требуется $O(nm)$ памяти. А как свести это количество к линейному?

Родственное расстояние

Фредерик Дамерау показал, что 80% ошибок при наборе текста человеком являются транспозициями.

Родственное расстояние

Фредерик Дамерау показал, что 80% ошибок при наборе текста человеком являются транспозициями. Если добавить четвертую операцию к уже определённым:

4. транспозиция символов: a**bc**d → a**cb**d,

то получим расстояние Дамерау–Левенштейна.

Вопросы?

Улучшаем потребление памяти

Пусть E — матрица, аналогичная D, только считаем расстояние не между префиксами строк, а между их суффиксами.

Улучшаем потребление памяти

Пусть E — матрица, аналогичная D, только считаем расстояние не между префиксами строк, а между их суффиксами.

Разобьём строку s на две с длинами $\frac{n}{2}$.

Улучшаем потребление памяти

Пусть E — матрица, аналогичная D, только считаем расстояние не между префиксами строк, а между их суффиксами.

Разобьём строку s на две с длинами $\frac{n}{2}$.

Для левой половины насчитаем D, для правой — E.

Улучшаем потребление памяти

Пусть E — матрица, аналогичная D, только считаем расстояние не между префиксами строк, а между их суффиксами.

Разобьём строку s на две с длинами $\frac{n}{2}$.

Для левой половины насчитаем D, для правой — E.

Переберём все разбиения строки t на две части и найдём минимальное $D(\frac{n}{2}, i) + E(\frac{n}{2}, m - i)$. Получается, что левую половину s мы превращаем в левую половину t оптимальным образом (то же самое верно и для правых половин).

Улучшаем потребление памяти

Пусть E — матрица, аналогичная D, только считаем расстояние не между префиксами строк, а между их суффиксами.

Разобьём строку s на две с длинами $\frac{n}{2}$.

Для левой половины насчитаем D, для правой — E.

Переберём все разбиения строки t на две части и найдём минимальное $D(\frac{n}{2}, i) + E(\frac{n}{2}, m - i)$. Получается, что левую половину s мы превращаем в левую половину t оптимальным образом (то же самое верно и для правых половин).

А теперь рекурсивно ищем решение для левых и правых половин строк s и t. После этого объединим полученные редакционные предписания.

Улучшаем потребление памяти

Пусть E — матрица, аналогичная D, только считаем расстояние не между префиксами строк, а между их суффиксами.

Разобьём строку s на две с длинами $\frac{n}{2}$.

Для левой половины насчитаем D, для правой — E.

Переберём все разбиения строки t на две части и найдём минимальное $D(\frac{n}{2}, i) + E(\frac{n}{2}, m - i)$. Получается, что левую половину s мы превращаем в левую половину t оптимальным образом (то же самое верно и для правых половин).

А теперь рекурсивно ищем решение для левых и правых половин строк s и t. После этого объединим полученные редакционные предписания.

Легко показать, что такой алгоритм требует $O(nm)$ операций и $O(n + \frac{n}{2} + \frac{n}{4} + \cdots) = O(n)$ памяти.