Edit distance

- An **alignment**, or matched up, of two strings is simply a way of writing the strings one above the other.

  example: alignments of "`SNOWY`*" and "*`SUNNY`*":*

  ```
  s - n o w y          - s n o w - y
  s u n n - y          s u n - - n y
  ```

  "–" indicates a "gap"

- The **cost** of an alignment is the number of columns in which the letters differ.

  example: alignments of "`SNOWY`*" and "*`SUNNY`*":*

  ```
  s - n o w y          - s n o w - y
  s u n n - y          s u n - - n y
   cost = 3              cost = 5
  ```

- **Edit distance** between two strings is the **minimum cost** of their alignment, i.e., *the best possible alignment*

Edit distance

- Given strings $x[1 \cdots m]$ and $y[1 \cdots n]$. Define

$$e(m, n) = \text{the edit distance between } x \text{ and } y$$

  Our objective is to compute $e(m, n)$ efficiently
- *Subproblem*:

$$\text{edit distance } e(i, j) \text{ between } x[1 \cdots i] \text{ and } y[1 \cdots j]$$

- How to express $e(i, j)$ in terms of its subproblems, *recursively*?
- **key observation:** the rightmost column of an alignment of $x[1 \cdots i]$ and $y[1 \cdots j]$ can only be one of the following three cases:

Case 1		Case 2		Case 3
$x[i]$	or	$-$	or	$x[i]$
$-$		$y[j]$		$y[j]$

Edit distance

- By the above key observation, then

$$e(i,j) = \min\{\underbrace{1 + e(i-1,j)}_{\text{case 1}},\ \underbrace{1 + e(i,j-1)}_{\text{case 2}},\ \underbrace{\mathrm{diff}(i,j) + e(i-1,j-1)}_{\text{case 3}}\}$$

where

$$\mathrm{diff}(i,j) = \begin{cases} 0 & \text{if } x[i] = y[j] \\ 1 & \text{if } x[i] \neq y[j] \end{cases}$$

- Question: how to find the corresponding optimal alignment?

Edit distance

- The answers to all the subproblems $e(i, j)$ form a two-dimensional table, and the final answer (our objective) is at $e(m, n)$.
- Initialization:

$$
\begin{aligned}
e(0,0) &= 0; \\
e(i,0) &= i \text{ for } i = 1, \ldots, m \\
e(0,j) &= j \text{ for } j = 1, \ldots, n
\end{aligned}
$$