Edit Distance

The Levenshtein (or edit) distance between strings s and t is the least cost sequence of edit commands that transform s to t. The possible edit commands are:

- **Copy** a character from s to t (cost = 0).
- **Delete** a character from s (cost = 1).
- **Insert** a character into t (cost = 1).
- **Substitute** one character for another (cost = 1).

Examples of the edit commands:

- **Delete**. If s is `Joey` and t is `Joe`, the least cost transformation is to delete `y` (and copy all other characters).
- **Insert** a character into t. If s is `Hilary` and t is `Hillary`, the least cost transformation is to insert an `l`.
- **Substitute** one character for another. If s is `Smyth` and t is `Smith`, the least cost transformation is to substitute, replacing `y` with `i`.

Computing Distance

Given two strings, s and t, we can contemplate the least cost edit distance between any two *substrings* of s and t. Let $cost(i, j)$ be the least cost transformation of $s[:i]$ to $t[:j]$. The ultimate goal is to determine $cost(m, n)$ where m and n are the lengths of s and t respectively.

Key idea In computing the edit distance between $s[:i]$ and $t[:j]$, we can focus on the effect of the most recent edit command. There are four possibilities. If the last edit was...

1. **Copy** This edit has cost 0 so $cost(i, j) = cost(i-1, j-1)$. Note: a copy is only possible when $s[i] = t[j]$.
2. **Substitute** This edit has cost 1 so $cost(i, j) = 1+cost(i-1, j-1)$. Note: a substitution is only possible when $s[i] \neq t[j]$. Example: suppose $i = j = 3$,

   ```
      0123
   s = _ABC
   t = _ABD
   ```

   (In the examples, we add `_` to the beginning of each string. This is a special start symbol that is always matched.)
3. **Delete** This means $s[i]$ is deleted. It also implies that $s[:i-1]$ is aligned with $t[:j]$. This edit has cost 1 and so $cost(i, j) = 1 + cost(i-1, j)$. Example: suppose $i = 4$ and $j = 3$,

```
    01234
s = _ABCD
t = _ABC
```

4. **Insert** This means $t[j]$ is inserted. It also implies that $s[:i]$ is aligned with $t[:j-1]$. This edit has cost 1 and so $cost(i,j) = 1 + cost(i,j-1)$. Example: suppose $i = 3$ and $j = 4$,

```
    01234
s = _ABC
t = _ABCD
```

Of the above possibilities, *we can choose whichever one is smallest.*

Putting it all together To compute the edit distance between s and t, we need to compute $cost(m,n)$. We build this solution up, starting at $cost(0,0)$ and working our way up. We can store $cost(i,j)$ in a matrix of size $m \times n$. All we need when computing $cost(i,j)$ is $cost(i-1,j-1)$, $cost(i-1,j)$, $cost(i,j-1)$.

Here is an example matrix where s is 'xyzABC' and t is 'ABC':

```
   _  A  B  C
_  0, 1, 2, 3
x  1, 1, 2, 3
y  2, 2, 2, 3
z  3, 3, 3, 3
A  4, 3, 4, 4
B  5, 4, 3, 4
C  6, 5, 4, 3
```