Lecture 25 - Edit Distance

Eric A. Autry

Edit Distance (ED)

Objective: turn the string `'spam'` into the string `'slime'` using as few insertions, deletions, and substitutions as possible.

```
'spam' ->    (insert 'e')     -> 'spame'
       -> (sub 'i' for 'a') -> 'spime'
       -> (sub 'l' for 'p') -> 'slime'
```

How about `'libate'` to `'flub'`?

```
'libate' ->    (insert 'f')     -> 'flibate'
         ->    (delete 'e')     -> 'flibat'
         ->    (delete 't')     -> 'fliba'
         ->    (delete 'a')     -> 'flib'
         -> (sub 'u' for 'i') -> 'flub'
```

Application: genetics

Edit Distance (ED)

Idea: compare the last letter in each string and use recursion.

Base Case: what if one of the strings is empty?

If the strings aren't empty, what are **all** of the possible situations?

- The last letters are the same, so no editing necessary. Move on to the next letter! (Use-it)
  Specific example 1: `S1 = aaab` and `S2 = baab`
- They are not the same, and it's best to delete. (Lose-it #1)
  Specific example 2: `S1 = aaab` and `S2 = aaa`
- They are not the same, and it's best to insert. (Lose-it #2)
  Specific example 3: `S1 = aaa` and `S2 = aaab`
- They are not the same, and it's best to sub. (Lose-it #3)
  Specific example 4: `S1 = aaab` and `S2 = aaaa`

We will consider **all** of these possibilities, and keep whichever gives us the smallest ED!

Edit Distance (ED)

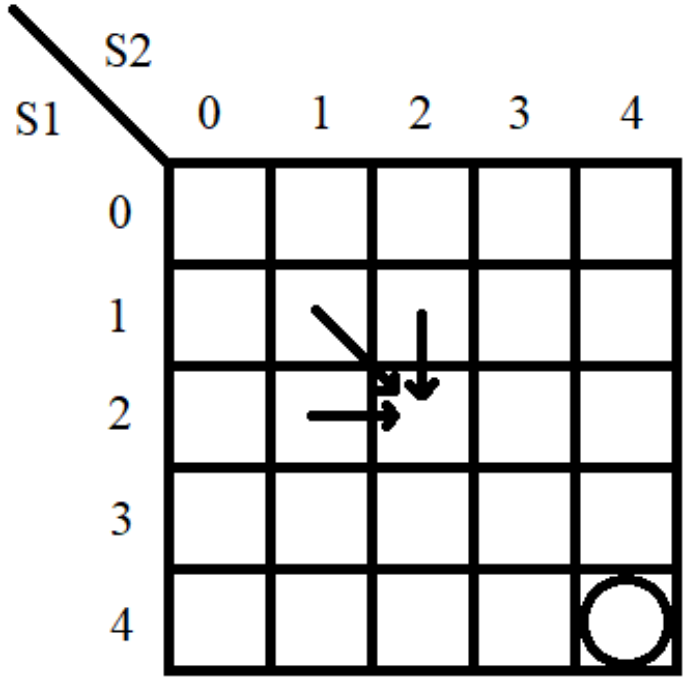
```
def ED(S1, S2, i, j):
    # Returns the ED for S1[0:i+1] and S2[0:j+1].

    # Base Case (no letters left to consider).
    if (i < 0) or (j < 0):
        return max(i+1, j+1) # length = index+1

    # Use It (they match so no edit needed).
    elif S1[i] == S2[j]:
        return ED(S1, S2, i-1, j-1)

    # Lose It (they didn't match so edit).
    else:
        A = ED(S1, S2, i-1, j) # Delete from S1
        B = ED(S1, S2, i, j-1) # Insert (match S2)
        C = ED(S1, S2, i-1, j-1) # Sub (match both)
        return 1 + min(A, B, C)
```

- Note for the 'Sub' case, they now match. So no more editing necessary and we can move on to the next letter!

Edit Distance (ED)

For ED, how many different recursive calls are made? $n \times m$

So store the results in an $n \times m$ DP table.



Note that the recursive calls come from above or to the left.

- Fill the table from top left to bottom right.
- Start with first row/column (base case).
- Then fill in diagonals (or 2nd row/col, then 3rd row/col, etc).
- $O(n \cdot m)$

Edit Distance (ED)

Ex: `ED('spam', 'pims')`

One possible way to fill in the table is:

	''	p	i	m	s
''	0	1	2	3	4
s	1	1	2	3	3
p	2	1	2	3	4
a	3	2	2	3	4
m	4	3	3	2	3

	''	p	i	m	s
''	0	1	2	3	4
s	1	D	L	L	D
p	2	D	L	L	L
a	3	U	D	L	L
m	4	U	U	D	L

The rules for reconstructing the solution are:

- If top row: insert the remaining letters of `'pims'`.
- If first column: delete the preceding letters of `'spam'`.
- L: insert the letter for this column
- U: delete the letter for this row
- D (+1): change letter for this row to the one for this column
- D (+0): match, no edit required

Edit Distance (ED)

Ex: `ED('spam', 'pims')`

	''	p	i	m	s
''	0	1	2	3	4
s	1	1	2	3	3
p	2	1	2	3	4
a	3	2	2	3	4
m	4	3	3	2	3

	''	p	i	m	s
''	0	1	2	3	4
s	1	D	L	L	D
p	2	D	L	L	L
a	3	U	D	L	L
m	4	U	U	D	L

- Start at bottom right.
- See L, insert `'s'` at the end. Move left.
- See D $(+0)$, match the `'m'`. Move diagonally.
- See D $(+1)$, change `'a'` into `'i'`. Move diagonally.
- See D $(+0)$, match the `'p'`. Move diagonally.
- See 1, base case in first column: remove the leading `'s'`.

Edit Distance (ED)

Ex: `ED('spam', 'pims')`

- Start at bottom right.
- See L, insert `'s'` at the end. Move left.
- See D $(+0)$, match the `'m'`. Move diagonally.
- See D $(+1)$, change `'a'` into `'i'`. Move diagonally.
- See D $(+0)$, match the `'p'`. Move diagonally.
- See 1, base case in first column: remove the leading `'s'`.

Working backwards through the string `'spam'`:

```
'spam' ->   (insert 's')     -> 'spams'
       -> (sub 'i' for 'a') -> 'spims'
       ->   (delete 's')     -> 'pims'
```

It took us 3 edits, which is what the DP table told us!