Porter Stemmer

The Porter Stemmer (Porter, 1980)

- A simple rule-based algorithm for stemming
- An example of a HEURISTIC method
- Based on rules like:
  - ATIONAL -> ATE (e.g., *relational* -> *relate*)
- The algorithm consists of seven sets of rules, applied in order

The Porter Stemmer: definitions

- Definitions:
  - CONSONANT: a letter other than A, E, I, O, U, and Y preceded by consonant
  - VOWEL: any other letter
- With this definition, all words are of the form:

  $(C)(VC)^{m}(V)$

  C=string zero or more consonants

  V=string of one or more vowels
- E.g.,
  - Troubles
  - C V CVC

The Porter Stemmer: rule format

- The rules are of the form:

  (condition) S1 -> S2

  Where S1 and S2 are suffixes

- Conditions:

m	The measure of the stem
*S	The stem ends with S
v	The stem contains a vowel
*d	The stem ends with a double consonant
*o	The stem ends in CVC (second C not W, X, or Y)

The Porter Stemmer: Step 1

1. SSES -> SS
    1. *expresses* -> express
2. IES -> I
    1. *ponies -> poni*
    2. *ties -> ti*
3. SS -> SS
    1. *process -> process*
4. S -> ϵ
    1. *cats -> cat*

The Porter Stemmer: Step 2a (past tense, progressive)

1. (m>1) EED -> EE
   1. Condition verified: *agreed -> agree*
   2. Condition not verified: *feed -> feed*
2. (*V*) ED -> є
   1. Condition verified: *plastered -> plaster*
   2. Condition not verified: *bled -> bled*
3. (*V*) ING -> є
   - Condition verified: *motoring -> motor*
   - Condition not verified: *sing -> sing*

The Porter Stemmer: Step 2b (cleanup)

- (These rules are ran if second or third rule in 2a apply)

4. AT-> ATE
   - *conflat(ed) -> conflate*
5. BL -> BLE
   - *Troubl(ing) -> trouble*
6. (*d & ! (*L or *S or *Z)) -> single letter
   - Condition verified: *hopp(ing) -> hop, tann(ed) -> tan*
   - Condition not verified: *fall(ing) -> fall*
7. (m>1 & *o) -> E
   - Condition verified: *fil(ing) -> file*
   - Condition not verified: *fail -> fail*

The Porter Stemmer: Steps 3 and 4

- Step 3: Y Elimination (*V*) Y -> I
  - Condition verified: *happy -> happi*
  - Condition not verified: *sky -> sky*
- Step 4: Derivational Morphology, I

8. (m>0) ATIONAL -> ATE
   - *Relational -> relate*
9. (m>0) IZATION -> IZE
   - *generalization-> generalize*
10. (m>0) BILITY-> BLE
    - *sensibility -> sensible*

The Porter Stemmer: Steps 5 and 6

- Step 5: Derivational Morphology, II
  - (m>0) ICATE -> IC
    - *triplicate -> triplic*
  - (m>0) FUL -> ϵ
    - *hopeful -> hope*
  - (m>0) NESS -> ϵ
    - *goodness -> good*
- Step 6: Derivational Morphology, III
  - (m>0) ANCE -> ϵ
    - *allowance-> allow*
  - (m>0) ENT -> ϵ
    - *dependent-> depend*
  - (m>0) IVE -> ϵ
    - *effective -> effect*
  - (m>0) IZE -> ϵ
    - *generalize -> general*
  - (m>0) ANT -> ϵ
    - *reluctant-> reluct*
  - (m>0) r -> ϵ
    - *computer -> compute*

The Porter Stemmer: Step 7 (cleanup)

- Step 7a
  - (m>1) E -> ϵ
    - *probate -> probat*
  - (m>1 & !*o) NESS -> ϵ
    - *goodness -> good*
- Step 7b
  - (m>1 & *d & *L) -> single letter
    - Condition verified*: controll -> control*
    - Condition not verified*: roll -> roll*

Examples

- *computers*
  - Step 1, Rule 4: -> *computer*
  - Step 6, Rule 4: -> *compute*
- *singing*
  - Step 2a, Rule 3: -> *sing*
- *controlling*
  - Step 2a, Rule 3: -> *controll*
  - Step 7b : -> *control*
- *generalizations*
  - Step 1, Rule 4: -> *generalization (noun)*
  - Step 4, Rule 9: -> *generalize (verb)*
  - Step 6, last rule: -> *general (adjective)*

Problems

- *elephants -> eleph*
  - Step 1, Rule 4: -> *elephant*
  - Step 6, Rule 7: -> *eleph*
- *doing - > do*
  - Step 2a, Rule 3: -> *do*

References

- The Porter Stemmer home page (with the original paper and code): http://www.tartarus.org/~martin/PorterStemmer/
- Jurafsky and Martin, chapter 3.4
- The original paper: Porter, M.F., 1980, An algorithm for suffix stripping, *Program*, **14**(3) :130-137.

SPELL CHECKER

Detecting and Correcting Spelling Error

Detecting and Correcting Spelling Error

- Detection and correction of spelling errors is an integral part of modern word processors and search engines.
- It is important in correcting errors in the recognition of human printed or cursive handwriting as the user is writing.

Categorization

- Categorized into 3 broader problems , as per Kukich(1992):

1. **non-word error detection:** detecting spelling errors that result in non-words (like *graffe for giraffe).*
2. **isolated-word error correction:** correcting spelling errors that result in nonwords, for example correcting *graffe to giraffe, but looking only at the word in* isolation.
3. **context-dependent error detection and correction:** using the context to help detect and correct spelling errors even if they accidentally result in an actual REALWORD word of English (**real-word errors).** This can happen from typographical errors (insertion, deletion, transposition) which accidentally produce a real word (e.g., *there for three), or because the writer substituted the wrong spelling of a* homophone or near-homophone (e.g., *dessert for desert, or piece for peace).*

Error detection

- Detecting non-word errors is generally done by marking any word that is not found in a dictionary.
- **For example**, the misspelling ***graffe*** *would not occur in a dictionary.*

Error Correction

- Algorithms for isolated-word error correction operate by finding words which are the likely source of the error-ful form.
- **For example**, correcting the spelling error ***graffe*** *requires searching through all possible words like giraffe, graf, craft, grail,* etc, to pick the most likely source.
- To choose among these potential sources we need a **distance metric between the source and the surface error.**
- **Intuitively, *giraffe is a more* likely source** than ***grail*** *for **graffe**, because **giraffe is closer in spelling to graffe** than **grail is to graffe**.*

MINIMUM EDIT DISTANCE (Wagner and Fischer 1974)

- The non-probabilistic algorithm for this solution is the **minimum edit distance algorithm.**
- **The distance between** two strings is a measure of how alike two strings are to each other.
- The minimum edit distance between two strings is the minimum number of editing operations (insertion, deletion, substitution) needed to transform one string into another.

Minimum Edit Distance

- For example the gap between the words intention and execution is five operations, shown in Fig. 3.23 as an alignment between the two strings.
- Given two sequences, an alignment is a correspondence between substrings of the two sequences. Thus I aligns with the empty string, N with E, T with X, and so on.
- Beneath the aligned strings is another representation; a series of symbols expressing an operation list for converting the top string into the bottom string; d for deletion, s for substitution, i for insertion.

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Figure 3.23 Representing the minimum edit distance between two strings as an **alignment**. The final row gives the operation list for converting the top string into the bottom string: d for deletion, s for substitution, i for insertion.

MINIMUM EDIT DISTANCE

- Cost or weight can be assigned to each of these operations.
- The **Levenshtein distance** between two sequences is the simplest weighting factor in which each of the three operations has a cost of 1 (Levenshtein, 1966).
- Thus the Levenshtein distance between *intention and execution is 5.*
- Levenshtein also proposed an alternate version of his metric in which each insertion or deletion has a cost of one, and substitutions are not allowed (equivalent to allowing substitution, but giving each substitution a cost of 2, since any substitution can be represented by one insertion and one deletion).
- Using this version, the Levenshtein distance between *intention and execution is 8.*

MINIMUM EDIT DISTANCE

- The minimum edit distance is computed by **dynamic programming.**
- The intuition of a dynamic programming problem is that a large problem can be solved by properly combining the solutions to various sub-problems.

How to find the Min Edit Distance?

- Searching for a path (sequence of edits) from the start string to the final string:
  - **Initial state**: the word we're transforming
  - **Operators**: insert, delete, substitute
  - **Goal state**: the word we're trying to get to
  - **Path cost**: what we want to minimize: the number of edits

Defining Min Edit Distance

- For two strings
  - X of length n
  - Y of length m
- We define $D(i, j)$
  - the edit distance between $X[1..i]$ and $Y[1..j]$
    - i.e., the first i characters of X and the first j characters of Y
  - The edit distance between X and Y is thus $D(n,m)$

MINIMUM EDIT DISTANCE

- Dynamic programming algorithms work by creating a distance matrix with one column for each symbol in the source sequence and one row for each symbol in the target sequence (i.e., source along the top, target along the side).
- For minimum edit distance, this matrix is the edit-distance matrix. Each cell edit-distance[i, j] contains the distance between the first i characters of the target and the first j characters of the source.
- Each cell can be computed as a simple function of the surrounding cells; thus starting from the beginning of the matrix it is possible to fill in every entry.
- The value in each cell is computed by taking the minimum of the three possible paths through the matrix which arrive there:

$$distance[i,j] = \min \begin{cases} distance[i-1,j] + \text{del-cost}(source_{i-1}) \\ distance[i-1,j-1] + \text{subst-cost}(source_{i-1}, target_{j-1}) \\ distance[i,j-1] + \text{ins-cost}(target_{j-1})) \end{cases}$$

Min-Edit distance Algorithm

```
function MIN-EDIT-DISTANCE(target, source) returns min-distance
    m ← LENGTH(target)
    n ← LENGTH(source)
    Create a distance matrix -distance[n+1, m+1]
    Initialize the zeroth row and column to be the distance
    from empty string
      distance[0,0] = 0
      for each row i from 1 to n do
         distance[i, 0] ← distance[i-1, 0] + ins-cost(target[i])
         for each column j from 1 to m do
         distance[0, j] ← distance[0, j-1] + del-cost(source[j])
     for each row i from 1 to n do
        for each column j from 1 to m do
        if ( i != j)
           distance[i, j] ← MIN (distance[i-1, j] + ins-cost(target_j),
                     distance[i-1, j-1] + subst-cost(source_i , target_j),
                     distance[i, j-1] + del-cost(source_j) )
      distance[i, j] ← distance[i-1, j-1]
    return distance[n, m]
```

Example:

- Convert string **abcdef** to string **azced**

	#	a	b	c	d	e	f
#	0	1	2	3	4	5	6
a	1						
z	2						
c	3						
e	4						
d	5						

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + 1 \end{cases}$$

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
T		#	a	b	c	d	e	f
a	#							
r	a							
g	z							
e	c							
t	e							
↓	d							

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0						
	a							
	z							
	c							
	e							
	d							

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1					
	a							
	z							
	c							
	e							
	d							

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →

Target ↓

	#	a	b	c	d	e	f
#	0	1	2				
a							
z							
c							
e							
d							

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3			
	a							
	z							
	c							
	e							
	d							

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

	Source →							
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4		
r	a							
g	z							
e	c							
t	e							
↓	d							

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	
	a							
	z							
	c							
	e							
	d							

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4	5	6
r	a							
g	z							
e	c							
t	e							
↓	d							

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1						
	z							
	c							
	e							
	d							

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

				Source →				
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4	5	6
r	a	1						
g	z	2						
e	c							
t	e							
↓	d							

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

				Source →				
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4	5	6
r	a	1						
g	z	2						
e	c	3						
t	e							
↓	d							

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4	5	6
r	a	1						
g	z	2						
e	c	3						
t	e	4						
↓	d							

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

		Source →						
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4	5	6
r	a	1						
g	z	2						
e	c	3						
t	e	4						
↓	d	5						

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

	Source →							
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4	5	6
r	a	1	0					
g	z	2						
e	c	3						
t	e	4						
↓	d	5						

Source: a

Target : a

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1				
	z	2						
	c	3						
	e	4						
	d	5						

Source: a b

Target : a

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

				Source →				
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4	5	6
r	a	1	0	1	2			
g	z	2						
e	c	3						
t	e	4						
↓	d	5						

Source: abc

Target : a

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		**#**	**a**	**b**	**c**	**d**	**e**	**f**
	#	**0**	**1**	**2**	**3**	**4**	**5**	**6**
	a	**1**	**0**	**1**	**2**	**3**		
	z	**2**						
	c	**3**						
	e	**4**						
	d	**5**						

Source: abcd

Target : a

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
T		**#**	**a**	**b**	**c**	**d**	**e**	**f**
a	**#**	**0**	**1**	**2**	**3**	**4**	**5**	**6**
r	**a**	**1**	**0**	**1**	**2**	**3**	**4**	
g	**z**	**2**						
e	**c**	**3**						
t	**e**	**4**						
↓	**d**	**5**						

Source: abcde

Target : a

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2						
	c	3						
	e	4						
	d	5						

Source: abcdef

Target : a

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1					
	c	3						
	e	4						
	d	5						

Source: a

Target : az

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4	5	6
r	a	1	0	1	2	3	4	5
g	z	2	1	1				
e	c	3						
t	e	4						
↓	d	5						

Source: ab

Target : az

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2			
	c	3						
	e	4						
	d	5						

Source: abc

Target : az

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2	3		
	c	3						
	e	4						
	d	5						

Source: abcd

Target : az

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4	5	6
r	a	1	0	1	2	3	4	5
g	z	2	1	1	2	3	4	
e	c	3						
t	e	4						
↓	d	5						

Source: abcde

Target : az

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4	5	6
r	a	1	0	1	2	3	4	5
g	z	2	1	1	2	3	4	5
e	c	3						
t	e	4						
↓	d	5						

Source: abcdef

Target : az

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2	3	4	5
	c	3	2					
	e	4						
	d	5						

Source: a

Target : azc

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
T		**#**	**a**	**b**	**c**	**d**	**e**	**f**
a	**#**	**0**	**1**	**2**	**3**	**4**	**5**	**6**
r	**a**	**1**	**0**	**1**	**2**	**3**	**4**	**5**
g	**z**	**2**	**1**	**1**	**2**	**3**	**4**	**5**
e	**c**	**3**	**2**	**2**				
t	**e**	**4**						
↓	**d**	**5**						

Source: ab

Target : azc

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4	5	6
r	a	1	0	1	2	3	4	5
g	z	2	1	1	2	3	4	5
e	c	3	2	2	1			
t	e	4						
↓	d	5						

Source: abc

Target : azc

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

	Source →							
T		**#**	**a**	**b**	**c**	**d**	**e**	**f**
a	**#**	**0**	**1**	**2**	**3**	**4**	**5**	**6**
r	**a**	**1**	**0**	**1**	**2**	**3**	**4**	**5**
g	**z**	**2**	**1**	**1**	**2**	**3**	**4**	**5**
e	**c**	**3**	**2**	**2**	**1**	**2**		
t	**e**	**4**						
↓	**d**	**5**						

Source: abcd

Target : azc

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
T		**#**	**a**	**b**	**c**	**d**	**e**	**f**
a	**#**	**0**	**1**	**2**	**3**	**4**	**5**	**6**
r	**a**	**1**	**0**	**1**	**2**	**3**	**4**	**5**
g	**z**	**2**	**1**	**1**	**2**	**3**	**4**	**5**
e	**c**	**3**	**2**	**2**	**1**	**2**	**3**	
t	**e**	**4**						
↓	**d**	**5**						

Source: abcde

Target : azc

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

	Source →							
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2	3	4	5
	c	3	2	2	1	2	3	4
	e	4						
	d	5						

Source: abcdef

Target : azc

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2	3	4	5
	c	3	2	2	1	2	3	4
	e	4	3					
	d	5						

Source: a

Target : azce

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2	3	4	5
	c	3	2	2	1	2	3	4
	e	4	3	3				
	d	5						

Source: ab

Target : azce

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
T		#	a	b	c	d	e	f
a	#	0	1	2	3	4	5	6
r	a	1	0	1	2	3	4	5
g	z	2	1	1	2	3	4	5
e	c	3	2	2	1	2	3	4
t	e	4	3	3	2			
↓	d	5						

Source: abc

Target : azce

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		**#**	**a**	**b**	**c**	**d**	**e**	**f**
	#	**0**	**1**	**2**	**3**	**4**	**5**	**6**
	a	**1**	**0**	**1**	**2**	**3**	**4**	**5**
	z	**2**	**1**	**1**	**2**	**3**	**4**	**5**
	c	**3**	**2**	**2**	**1**	**2**	**3**	**4**
	e	**4**	**3**	**3**	**2**	**2**		
	d	**5**						

Source: abcd

Target : azce

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
T		**#**	**a**	**b**	**c**	**d**	**e**	**f**
a	**#**	**0**	**1**	**2**	**3**	**4**	**5**	**6**
r	**a**	**1**	**0**	**1**	**2**	**3**	**4**	**5**
g	**z**	**2**	**1**	**1**	**2**	**3**	**4**	**5**
e	**c**	**3**	**2**	**2**	**1**	**2**	**3**	**4**
t	**e**	**4**	**3**	**3**	**2**	**2**	**2**	
↓	**d**	**5**						

Source: abcde

Target : azce

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2	3	4	5
	c	3	2	2	1	2	3	4
	e	4	3	3	2	2	2	3
	d	5						

Source: abcdef

Target : azce

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		**#**	**a**	**b**	**c**	**d**	**e**	**f**
	#	**0**	**1**	**2**	**3**	**4**	**5**	**6**
	a	**1**	**0**	**1**	**2**	**3**	**4**	**5**
	z	**2**	**1**	**1**	**2**	**3**	**4**	**5**
	c	**3**	**2**	**2**	**1**	**2**	**3**	**4**
	e	**4**	**3**	**3**	**2**	**2**	**2**	**3**
	d	**5**	**4**					

Source: a

Target : azced

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2	3	4	5
	c	3	2	2	1	2	3	4
	e	4	3	3	2	2	2	3
	d	5	4	4				

Source: ab

Target : azced

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2	3	4	5
	c	3	2	2	1	2	3	4
	e	4	3	3	2	2	2	3
	d	5	4	4	3			

Source: abc

Target : azced

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2	3	4	5
	c	3	2	2	1	2	3	4
	e	4	3	3	2	2	2	3
	d	5	4	4	3	2		

Source: abcd

Target : azced

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

					Source →			
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2	3	4	5
	c	3	2	2	1	2	3	4
	e	4	3	3	2	2	2	3
	d	5	4	4	3	2	3	

Source: abcde

Target : azced

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2	3	4	5
	c	3	2	2	1	2	3	4
	e	4	3	3	2	2	2	3
	d	5	4	4	3	2	3	3

Source: abcdef

Target : azced

Example:

Given **source** string: **abcdef**

Convert into **target** string: **azced**

Source →								
Target ↓		#	a	b	c	d	e	f
	#	0	1	2	3	4	5	6
	a	1	0	1	2	3	4	5
	z	2	1	1	2	3	4	5
	c	3	2	2	1	2	3	4
	e	4	3	3	2	2	2	3
	d	5	4	4	3	2	3	3

Source: abcdef

Target : azced

Example 2:

- Find the minimum number of edit operations required to convert **intention to execution**

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	1	3							
X	2									
E	3									
C	4									
U	5									
T	6									
I	7									
O	8									
N	9									

Src: IN
Tar: E
Delete I, Subs:N ⟵ E

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	2	3	4	5	6	7	8	9	10
X	2									
E	3									
C	4									
U	5									
T	6									
I	7									
O	8									
N	9									

Src: INTENTION
Tar: E
Delete source

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	1	3	4	5	6	7	8	9	10
X	2	2								
E	3									
C	4									
U	5									
T	6									
I	7									
O	8									
N	9									

Src: I
Tar: EX
Delete I, Insert E, X

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	1	3							
X	2			5						
E	3									
C	4									
U	5									
T	6									
I	7									
O	8									
N	9									

Src: INT
Tar: EX
Delete I, Subs:T ← X

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	1	3							
X	2			5						
E	3				5					
C	4									
U	5									
T	6									
I	7									
O	8									
N	9									

Src: INTE
Tar: EXE
No subs cost

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	1	3							
X	2			5						
E	3				5					
C	4				6					
U	5									
T	6									
I	7									
O	8									
N	9									

Src: INTE
Tar: EXEC
Insert C

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	1	3							
X	2			5						
E	3				5					
C	4				6					
U	5									
T	6									
I	7									
O	8									
N	9									

Src: INTEN
Tar: EXECU
Subs:N $\leftarrow$ U

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	1	3							
X	2			5						
E	3				5					
C	4				6					
U	5					8				
T	6									
I	7									
O	8									
N	9									

Src: INTEN
Tar: EXECU
Subs:N ← U

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	1	3							
X	2			5						
E	3				5					
C	4				6					
U	5					8				
T	6						8			
I	7									
O	8									
N	9									

Src: INTENT
Tar: EXECUT
No Subs

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	1	3							
X	2			5						
E	3				5					
C	4				6					
U	5					8				
T	6						8			
I	7							8		
O	8									
N	9									

Src: INTENTI
Tar: EXECUTI
No Subs

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	1	3							
X	2			5						
E	3				5					
C	4				6					
U	5					8				
T	6						8			
I	7							8		
O	8								8	
N	9									

Src: INTENTIO
Tar: EXECUTIO
No Subs

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	1	3							
X	2			5						
E	3				5					
C	4				6					
U	5					8				
T	6						8			
I	7							8		
O	8								8	
N	9									8

Src: INTENTION
Tar: EXECUTION
No Subs

The Edit Distance Table

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Insertion cost=1
Deletion Cost=1
Substitution Cost=2

	#	I	N	T	E	N	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
E	1	1	3							
X	2			5						
E	3				5					
C	4				6					
U	5					8				
T	6						8			
I	7							8		
O	8								8	
N	9									8

$$\text{ptr}(i, j) = \begin{cases} \text{LEFT} & \text{deletion} \\ \text{UP} & \text{insertion} \\ \text{DIAG} & \text{substitution} \end{cases}$$

Example 3:

- Find the minimum number of edit operations required to convert **load to lord, graffe to giraffe**