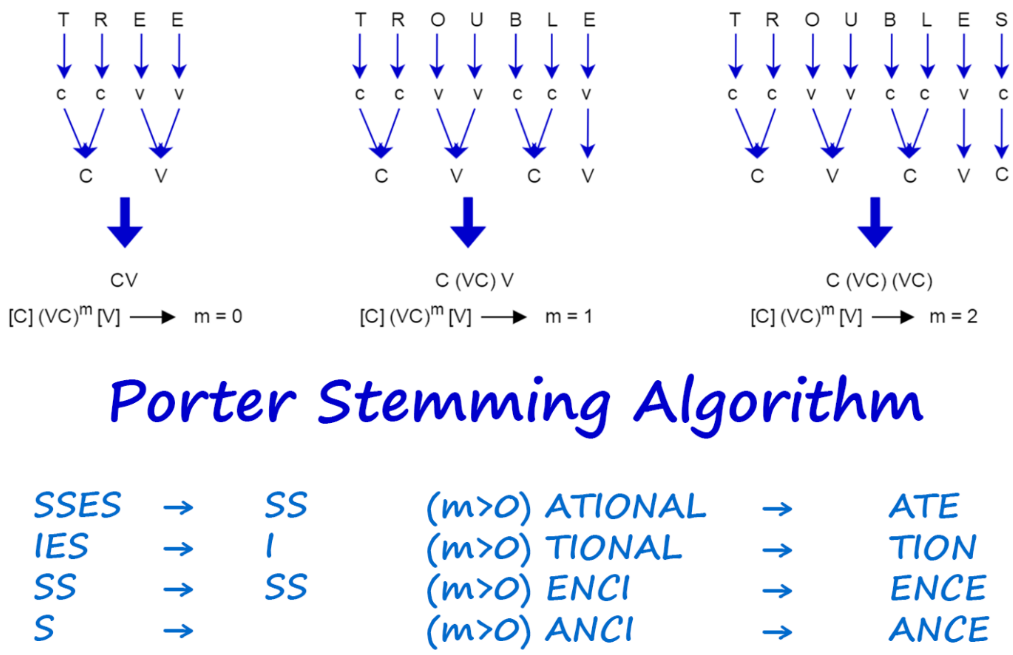
LAB 5 : Implement Porter stemmer algorithm

In linguistics, a stem is a part of a word that is common to all of its inflected variants. For example, "connect," "connected," "connection," and "connecting" share the stem "connect." Stemming is the process of reducing inflected or derived words to their word stem.

The Porter Stemming algorithm, developed by Martin F. Porter, is a widely used method to remove suffixes from English words and obtain their stems. This algorithm is particularly useful in Information Retrieval systems, where reducing terms to their stems can enhance efficiency in terms of space and time complexity.



SSES	→	SS	(m>0) ATIONAL	→	ATE
IES	→	I	(m>0) TIONAL	→	TION
SS	→	SS	(m>0) ENCI	→	ENCE
S	→		(m>0) ANCI	→	ANCE

Definitions

Consonants and Vowels:

- A consonant (c) is any letter other than a vowel (a, e, i, o, u) and "y" preceded by a consonant.
- A vowel (v) is any vowel character.
- Consecutive consonants are denoted by C, and consecutive vowels by V.

Word Forms: Words or parts of words can have one of the following forms:

- CVCV...C
- CVCV...V
- VCVC...C
- VCVC...V

Algorithm Steps

Step 1a

Replace suffixes according to specified rules:

- SSES → SS
- IES → I
- SS → SS
- S →

Step 1b

Apply additional suffix transformations if conditions are met:

- EED → EE (if m > 0)
- ED → (if stem contains a vowel)
- ING → (if stem contains a vowel)
- Transformations for specific suffixes: AT→ATE, BL→BLE, IZ→IZE, *d (double consonant) conditions

Step 1c

Further transformations:

- Y → I (if stem contains a vowel)

Step 2

Replace suffixes with specified alternatives:

- ATIONAL→ATE
- TIONAL→TION
- ENCI→ENCE
- ANCI→ANCE
- IZER→IZE
- and more...

Step 3

Additional suffix replacements:

- ICATE→IC
- ATIVE→
- ALIZE→AL
- and more...

Step 4

More suffix replacements based on conditions:

- AL→
- ANCE→
- ENCE→
- ER→
- IC→
- and more...

Step 5a

Final transformations:

- E→
- E (if m=1 and not ending cvc)

Step 5b

Final adjustments:

- (m > 1 and *d and *L)→ single letter

Example Inputs

Example 1

Input: MULTIDIMENSIONAL

- No matches in steps 1, 2, and 3.
- Step 4: Delete "AL" (since m > 1)
- No further changes in step 5. Output: MULTIDIMENSION → MULTIDIMENSION

Example 2

Input: CHARACTERIZATION

- Step 2: Replace "IZATION" with "IZE"
- Step 4: Delete "IZE" (since m > 1)
- No further changes in other steps. Output: CHARACTERIZATION → CHARACTERIZE → CHARACTER

The Porter Stemming algorithm iteratively applies these rules to reduce words to their stems, facilitating tasks such as information retrieval and natural language processing.

IMPLEMENTATION

In [1]:
```
import re
```

In [2]:
```python
def stemRule(word, rule, replacement=""):
    """
    Get word to stem.
    The rule that is a regular expression.
    And replacement is what we add to the
    word after removing the rule matched.
    """
    word = str.strip(str.lower(word))
    rule = str.lower(rule)
    replacement = str.lower(replacement)

    tmp = re.split(rule, word) if re.search(rule, word) is not None else ""
    tmp = "".join(list(filter(None,tmp)))
    if tmp:
        tmp += replacement
        return(tmp)
    else:
        return("")
```

In [3]:
```python
def calMeasure(word, suffix=""):
    """
    [C](VC)^m[V]
    Function that returns m
    http://facweb.cs.depaul.edu/mobasher/classes/CSC575/papers/porter-algori
    """
    word   = str.lower(word)
    suffix = str.lower(suffix)

    if suffix:
        word = stemRule(word, suffix)
        if not word:
            return 0
    m = 0

    # Removing [C]
    RC = re.split(r"^[^aeiou]*", word)
    RC = "".join(list(filter(None,RC)))


    # Removing [V]
    RV = re.split(r"[aeiou]*$", RC)
    RV = "".join(list(filter(None,RV)))


    # Counting VC pairs
    tmp = RV
    while(len(tmp)):
        tmp = re.split(r"^[aeiou]+", tmp)
        tmp = "".join(list(filter(None,tmp)))
        tmp = re.split(r"^[^aeiou]+", tmp)
        tmp = "".join(list(filter(None,tmp)))
        m += 1
    return(m)
```

```
In [4]: def step1a(word):
            """
            SSES -> SS                                   caresses  ->  caress
            IES  -> I                                    ponies    ->  poni
                                                         ties      ->  ti
            SS   -> SS                                   caress    ->  caress
            S    ->                                      cats      ->  cat
            """
            # SSES to SS
            stem = stemRule(word, r"sses$", "ss")
            if stem: return stem

            # IES to I
            stem = stemRule(word, r"ies$", "i")
            if stem: return stem

            # SS to SS
            stem = stemRule(word, r"ss$", "ss")
            if stem: return stem

            # S to ''
            stem = stemRule(word, r"s$", "")
            if stem: return stem

            return word
```

Conditions

*S - the stem ends with S (and similarly for the other letters).

v - the stem contains a vowel.

*d - the stem ends with a double consonant (e.g. -TT, -SS).

*o - the stem ends cvc, where the second c is not W, X or Y (e.g. -WIL, -HOP).

In [5]:

```python
def conditionS(word, suffix=""):
    """
    *S - the stem ends with S (and similarly for the other letters).
    """
    if suffix:
        word   = re.split(suffix, word)
        word = "".join(list(filter(None, word)))
    if re.search(r"s$", word) is not None:
        return True
    else:
        return False

def conditionVowel(word, suffix=""):
    """
    *v* - the stem contains a vowel.
    """
    if suffix:
        word   = re.split(suffix, word)
        word = "".join(list(filter(None, word)))
    if re.search(r"[aeiou]+", word) is not None:
        return True
    else:
        return False

def conditionDC(word, suffix=""):
    """
    *d - the stem ends with a double consonant (e.g. -TT, -SS).
    """
    if suffix:
        word   = re.split(suffix, word)
        word = "".join(list(filter(None, word)))
    V = ['a', 'e', 'i', 'o', 'u']
    consonant1 = word[-1]
    if consonant1 in V: return False
    consonant2 = word[-2]
    if consonant2 in V: return False
    if consonant1 == consonant2:
        return True
    else:
        return False

def conditionO(word, suffix=""):
    """
    *o - the stem ends cvc, where the second c is not W, X or Y (e.g. -WIL,
    """
    if suffix:
        word   = re.split(suffix, word)
        word = "".join(list(filter(None, word)))
    V = ['a', 'e', 'i', 'o', 'u']
    if re.search(r"([^aeiou][aeiou][b-df-hj-np-tvz])$", word) is not None:
        return True
    else:
        return False

def conditionL(word, suffix=""):
    """
    *L - Stem ends with L
    """
    if suffix:
        word   = re.split(suffix, word)
        word = "".join(list(filter(None, word)))
    if re.search(r"l$", word) is not None:
        return True
    else:
        return False

def conditionZ(word, suffix=""):
    """
    *Z - Stem ends with Z
    """
    if suffix:
        word   = re.split(suffix, word)
        word = "".join(list(filter(None, word)))
    if re.search(r"z$", word) is not None:
        return True
    else:
        return False

def conditionT(word, suffix=""):
    """
    *T - Stem ends with T
    """
    if suffix:
        word   = re.split(suffix, word)
        word = "".join(list(filter(None, word)))
    if re.search(r"T$", word) is not None:
        return True
    else:
        return False
```

```
In [6]: def innerStep1b(word):
            """
            AT -> ATE                       conflat(ed)  ->  conflate
            BL -> BLE                       troubl(ed)   ->  trouble
            IZ -> IZE                       siz(ed)      ->  size
            (*d and not (*L or *S or *Z))
               -> single letter
                                            hopp(ing)    ->  hop
                                            tann(ed)     ->  tan
                                            fall(ing)    ->  fall
                                            hiss(ing)    ->  hiss
                                            fizz(ed)     ->  fizz
            (m=1 and *o) -> E               fail(ing)    ->  fail
                                            fil(ing)     ->  file
            """
            # AT -> ATE
            stem = stemRule(word, r"at$", "ate")
            if stem: return stem

            # BL -> BLE
            stem = stemRule(word, r"bl$", "ble")
            if stem: return stem

            # IZ -> IZE
            stem = stemRule(word, r"iz$", "ize")
            if stem: return stem

            # (*d and not (*L or *S or *Z))
            if conditionDC(word) and not (conditionL(word) or conditionS(word) or co
                return word[:-1]
            elif conditionDC(word) and (conditionL(word) or conditionS(word) or cond
                return word

            # (m=1 and *o) -> E
            if (calMeasure(word)==1) and conditionO(word):
                return (word + "e")
            else:
                return word
```

In [7]:
```
def step1b(word):
    """
    (m>0) EED -> EE                   feed      ->  feed
                                      agreed    ->  agree
    (*v*) ED  ->                      plastered ->  plaster
                                      bled      ->  bled
    (*v*) ING ->                      motoring  ->  motor
                                      sing      ->  sing
    """
    # (m>0) EED -> EE
    if calMeasure(word, r"eed$") > 0:
        stem = stemRule(word, r"eed$", "ee")
        if stem: return stem
    elif stemRule(word, r"eed$", "ee"):
        return word


    # (*v*) ED  -> ''
    if conditionVowel(word, r"ed$"):
        stem = stemRule(word, r"ed$", "")
        if stem: return innerStep1b(stem)
    elif stemRule(word, r"ed$", ""):
        return word

    # (*v*) ING ->
    if conditionVowel(word, r"ing$"):
        stem = stemRule(word, r"ing$", "")
        if stem: return innerStep1b(stem)
    elif stemRule(word, r"ing$"):
        return word

    return word
```

In [8]:
```
def step1c(word):
    """
    (*v*) Y -> I                   happy        ->  happi
                                   sky          ->  sky
    """
    if conditionVowel(word, r"y$"):
        stem = stemRule(word, r"y$", "i")
        if stem: return stem
    elif stemRule(word, r"y$"):
        return word

    return word
```

In [9]:
```
def measureStem(word, re, replac, m=0):
    """
    m > 0 - default
    """
    if calMeasure(word, re) > m:
        stem = stemRule(word, re, replac)
        if stem: return stem
    elif stemRule(word, re, replac):
        # Catering for conditions where there is a match but no change takes
        return word
    else:
        return False
```

```
In [10]: def step2(word):
    """
    (m>0) ATIONAL ->  ATE           relational     ->  relate
    (m>0) TIONAL  ->  TION          conditional    ->  condition
                                    rational       ->  rational
    (m>0) ENCI    ->  ENCE          valenci        ->  valence
    (m>0) ANCI    ->  ANCE          hesitanci      ->  hesitance
    (m>0) IZER    ->  IZE           digitizer      ->  digitize
    (m>0) ABLI    ->  ABLE          conformabli    ->  conformable
    (m>0) ALLI    ->  AL            radicalli      ->  radical
    (m>0) ENTLI   ->  ENT           differentli    ->  different
    (m>0) ELI     ->  E             vileli        - >  vile
    (m>0) OUSLI   ->  OUS           analogousli    ->  analogous
    (m>0) IZATION ->  IZE           vietnamization ->  vietnamize
    (m>0) ATION   ->  ATE           predication    ->  predicate
    (m>0) ATOR    ->  ATE           operator       ->  operate
    (m>0) ALISM   ->  AL            feudalism      ->  feudal
    (m>0) IVENESS ->  IVE           decisiveness   ->  decisive
    (m>0) FULNESS ->  FUL           hopefulness    ->  hopeful
    (m>0) OUSNESS ->  OUS           callousness    ->  callous
    (m>0) ALITI   ->  AL            formaliti      ->  formal
    (m>0) IVITI   ->  IVE           sensitiviti    ->  sensitive
    (m>0) BILITI  ->  BLE           sensibiliti    ->  sensible
    """
    # (m>0) ATIONAL ->  ATE
    x = measureStem(word, r"ATIONAL$", "ATE")
    if x: return x

    # (m>0) TIONAL  ->  TION
    x = measureStem(word, r"TIONAL$", "TION")
    if x: return x

    # (m>0) ENCI    ->  ENCE
    x = measureStem(word, r"ENCI$", "ENCE")
    if x: return x

    # (m>0) ANCI    ->  ANCE
    x = measureStem(word, r"ANCI$", "ANCE")
    if x: return x

    # (m>0) IZER    ->  IZE
    x = measureStem(word, r"IZER$", "IZE")
    if x: return x

    # (m>0) ABLI    ->  ABLE
    x = measureStem(word, r"ABLI$", "ABLE")
    if x: return x

    # (m>0) ALLI    ->  AL
    x = measureStem(word, r"ALLI$", "AL")
    if x: return x

    # (m>0) ENTLI   ->  ENT
    x = measureStem(word, r"ENTLI$", "ENT")
    if x: return x

    # (m>0) ELI     ->  E
    x = measureStem(word, r"ELI$", "E")
    if x: return x

    # (m>0) OUSLI   ->  OUS
    x = measureStem(word, r"OUSLI$", "OUS")
    if x: return x
    # (m>0) IZATION ->  IZE
    x = measureStem(word, r"IZATION$", "IZE")
    if x: return x

    # (m>0) ATION   ->  ATE
    x = measureStem(word, r"ATION$", "ATE")
    if x: return x

    # (m>0) ATOR    ->  ATE
    x = measureStem(word, r"ATOR$", "ATE")
    if x: return x

    # (m>0) ALISM   ->  AL
    x = measureStem(word, r"ALISM$", "AL")
    if x: return x

    # (m>0) IVENESS ->  IVE
    x = measureStem(word, r"IVENESS$", "IVE")
    if x: return x

    # (m>0) FULNESS ->  FUL
    x = measureStem(word, r"FULNESS$", "FUL")
    if x: return x

    # (m>0) OUSNESS ->  OUS
    x = measureStem(word, r"OUSNESS$", "OUS")
    if x: return x

    # (m>0) ALITI   ->  AL
    x = measureStem(word, r"ALITI$", "AL")
    if x: return x

    # (m>0) IVITI   ->  IVE
    x = measureStem(word, r"IVITI$", "IVE")
    if x: return x

    # (m>0) BILITI  ->  BLE
    x = measureStem(word, r"BILITI$", "BLE")
    if x: return x

    return word
```

```
In [11]: def step3(word):
             """
             (m>0) ICATE ->  IC              triplicate     ->  triplic
             (m>0) ATIVE ->                  formative      ->  form
             (m>0) ALIZE ->  AL              formalize      ->  formal
             (m>0) ICITI ->  IC              electriciti    ->  electric
             (m>0) ICAL  ->  IC              electrical     ->  electric
             (m>0) FUL   ->                  hopeful        ->  hope
             (m>0) NESS  ->                  goodness       ->  good
             """
             # (m>0) ICATE ->  IC
             x = measureStem(word, r"ICATE$", "IC")
             if x: return x

             # (m>0) ATIVE ->
             x = measureStem(word, r"ATIVE$", "")
             if x: return x

             # (m>0) ALIZE ->  AL
             x = measureStem(word, r"ALIZE$", "AL")
             if x: return x

             # (m>0) ICITI ->  IC
             x = measureStem(word, r"ICITI$", "IC")
             if x: return x

             # (m>0) ICAL  ->  IC
             x = measureStem(word, r"ICAL$", "IC")
             if x: return x

             # (m>0) FUL   ->
             x = measureStem(word, r"FUL$", "")
             if x: return x

             # (m>0) NESS  ->
             x = measureStem(word, r"NESS$", "")
             if x: return x

             return word
```

```
In [12]: def step4(word):
             """
             (m>1) AL    ->                  revival        ->   reviv
             (m>1) ANCE  ->                  allowance      ->   allow
             (m>1) ENCE  ->                  inference      ->   infer
             (m>1) ER    ->                  airliner       ->   airlin
             (m>1) IC    ->                  gyroscopic     ->   gyroscop
             (m>1) ABLE  ->                  adjustable     ->   adjust
             (m>1) IBLE  ->                  defensible     ->   defens
             (m>1) ANT   ->                  irritant       ->   irrit
             (m>1) EMENT ->                  replacement    ->   replac
             (m>1) MENT  ->                  adjustment     ->   adjust
             (m>1) ENT   ->                  dependent      ->   depend
             (m>1 and (*S or *T)) ION ->     adoption       ->   adopt
             (m>1) OU    ->                  homologou      ->   homolog
             (m>1) ISM   ->                  communism      ->   commun
             (m>1) ATE   ->                  activate       ->   activ
             (m>1) ITI   ->                  angulariti     ->   angular
             (m>1) OUS   ->                  homologous     ->   homolog
             (m>1) IVE   ->                  effective      ->   effect
             (m>1) IZE   ->                  bowdlerize     ->   bowdler
             """
             # (m>1) AL    ->
             x = measureStem(word, r"AL$", "", 1)
             if x: return x

             # (m>1) ANCE  ->
             x = measureStem(word, r"ANCE$", "", 1)
             if x: return x

             # (m>1) ENCE  ->
             x = measureStem(word, r"ENCE$", "", 1)
             if x: return x

             # (m>1) ER    ->
             x = measureStem(word, r"ER$", "", 1)
             if x: return x

             # (m>1) IC    ->
             x = measureStem(word, r"IC$", "", 1)
             if x: return x

             # (m>1) ABLE  ->
             x = measureStem(word, r"ABLE$", "", 1)
             if x: return x

             # (m>1) IBLE  ->
             x = measureStem(word, r"IBLE$", "", 1)
             if x: return x

             # (m>1) ANT   ->
             x = measureStem(word, r"ANT$", "", 1)
             if x: return x

             # (m>1) EMENT ->
             x = measureStem(word, r"EMENT$", "", 1)
             if x: return x

             # (m>1) MENT  ->
             x = measureStem(word, r"MENT$", "", 1)
             if x: return x

             # (m>1) ENT   ->
```

```
    x = measureStem(word, r"ENT$", "", 1)
    if x: return x

    # (m>1 and (*S or *T)) ION ->
    x = measureStem(word, r"ION$", "", 1)
    if x and (conditionS(word) or conditionT(word)):
        return x
    elif x:
        return x

    # (m>1) OU    ->
    x = measureStem(word, r"OU$", "", 1)
    if x: return x

    # (m>1) ISM   ->
    x = measureStem(word, r"ISM$", "", 1)
    if x: return x

    # (m>1) ATE   ->
    x = measureStem(word, r"ATE$", "", 1)
    if x: return x

    # (m>1) ITI   ->
    x = measureStem(word, r"ITI$", "", 1)
    if x: return x

    # (m>1) OUS   ->
    x = measureStem(word, r"OUS$", "", 1)
    if x: return x

    # (m>1) IVE   ->
    x = measureStem(word, r"IVE$", "", 1)
    if x: return x

    # (m>1) IZE   ->
    x = measureStem(word, r"IZE$", "", 1)
    if x: return x

    return word
```

In [13]:
```
def step5a(word):
    """
    (m>1) E     ->                  probate        ->  probat
                                    rate           ->  rate
    (m=1 and not *o) E ->           cease          ->  ceas
    """
    # (m>1) E     ->
    x = measureStem(word, r"E$", "", 1)
    if x:
        if x is not word: return x

    # (m=1 and not *o) E ->
    if (calMeasure(word)==1) and (not conditionO(word)):
        x = measureStem(word, r"E$", "")
        if x: return x

    return word
```

In [14]:
```
def step5b(word):
    """
    (m > 1 and *d and *L) -> single letter
                                   controll       ->  control
                                   roll           ->  roll
    """
    x = measureStem(word, r"\w$", "", 1)
    if x and (conditionDC(word) or conditionL(word)):
        return x
    elif x:
        return x
    else:
        return word
```

In [15]:
```
def porterStemmer(word):
    word = step1c(step1b(step1a(word)))
    print(word)
    word = step2(word)
    print(word)
    word = step3(word)
    print(word)
    word = step4(word)
    print(word)
    word = step5b(step5a(word))
    return word
```

In [16]:
```
(porterStemmer("filing"))
```

```
file
file
file
file
```

Out[16]:
```
'fil'
```

In [17]:
```
(porterStemmer("Running"))
```

```
run
run
run
run
```

Out[17]:
```
'run'
```