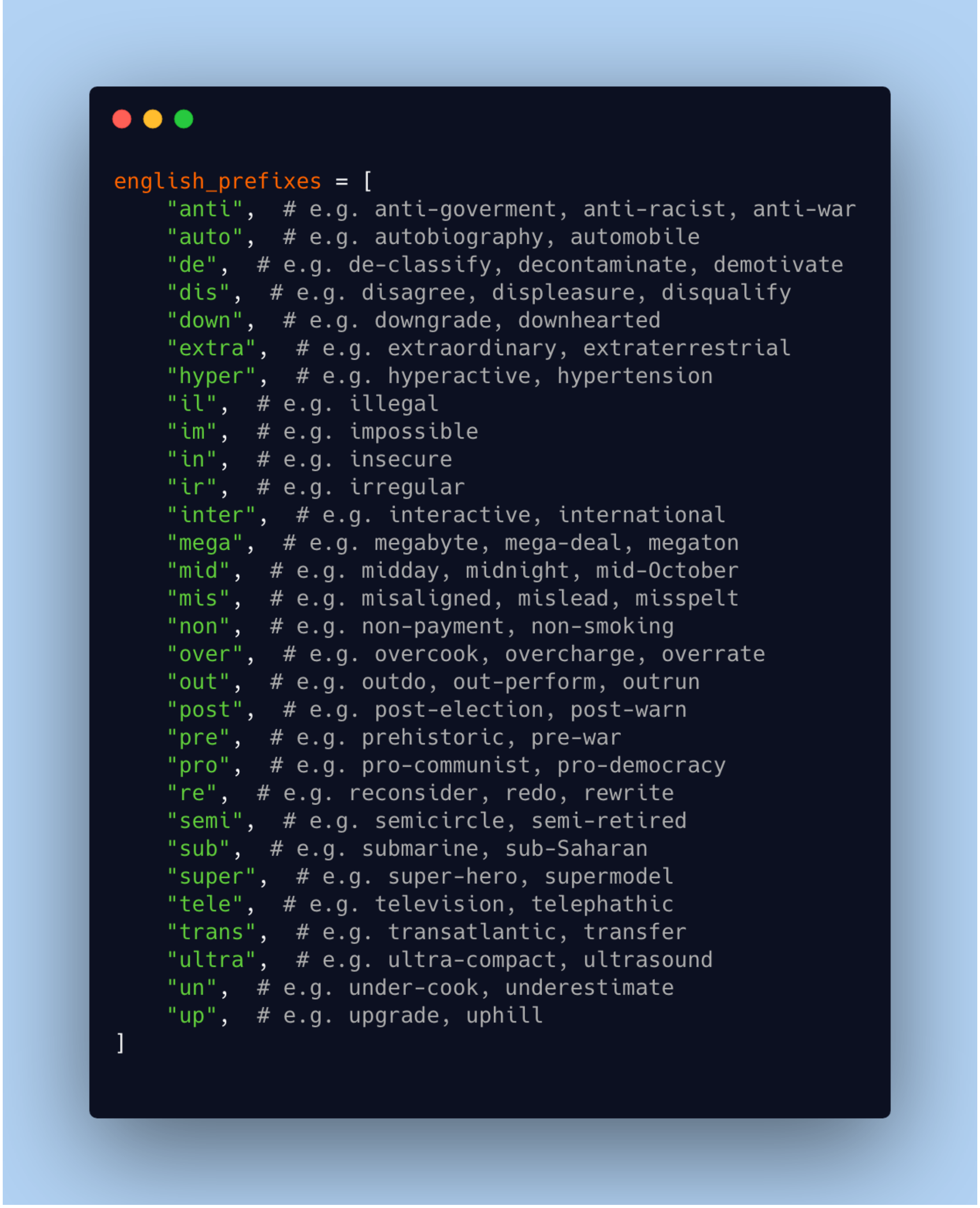
NLP Lab Assignment 3

Name : Anjishnu Mukherjee
Registration Number : B05-511017020
Exam Roll Number : 510517086
Email : 511017020.anjishnu@students.iiests.ac.in

Prefix Stemming

I take a list of common english prefixes from https://dictionary.cambridge.org/grammar/british-grammar/word-formation/prefixes .



```
english_prefixes = [
    "anti",  # e.g. anti-goverment, anti-racist, anti-war
    "auto",  # e.g. autobiography, automobile
    "de",  # e.g. de-classify, decontaminate, demotivate
    "dis",  # e.g. disagree, displeasure, disqualify
    "down",  # e.g. downgrade, downhearted
    "extra",  # e.g. extraordinary, extraterrestrial
    "hyper",  # e.g. hyperactive, hypertension
    "il",  # e.g. illegal
    "im",  # e.g. impossible
    "in",  # e.g. insecure
    "ir",  # e.g. irregular
    "inter",  # e.g. interactive, international
    "mega",  # e.g. megabyte, mega-deal, megaton
    "mid",  # e.g. midday, midnight, mid-October
    "mis",  # e.g. misaligned, mislead, misspelt
    "non",  # e.g. non-payment, non-smoking
    "over",  # e.g. overcook, overcharge, overrate
    "out",  # e.g. outdo, out-perform, outrun
    "post",  # e.g. post-election, post-warn
    "pre",  # e.g. prehistoric, pre-war
    "pro",  # e.g. pro-communist, pro-democracy
    "re",  # e.g. reconsider, redo, rewrite
    "semi",  # e.g. semicircle, semi-retired
    "sub",  # e.g. submarine, sub-Saharan
    "super",  # e.g. super-hero, supermodel
    "tele",  # e.g. television, telephathic
    "trans",  # e.g. transatlantic, transfer
    "ultra",  # e.g. ultra-compact, ultrasound
    "un",  # e.g. under-cook, underestimate
    "up",  # e.g. upgrade, uphill
]
```

Then for each input word, if the word starts with a prefix from this list, I remove the prefix temporarily from the word and check if the word without the prefix is a proper english word. To do so, I check if the word is present in the nltk wordnet corpus. If it is, then we can safely remove the prefix and stem the word. Otherwise, the prefix is not stemmed. (eg. ***pro***gramming) The output from the prefix stemming process is passed to the Suffix Stemmer.

Suffix Stemming

The rules used for suffix stemming are listed below.
These rules are checked from top to bottom and performed in order when a match is found.

The word is first converted to a normalised representation of **[C](VC)^m[V]**, where C is a consonant sequence and V is a vowel sequence.

Condition	Rule	Example
1. Handle plurals and past participles		
Word ends with SSES, IES, SS, S	Replace SSES -> SS IES -> I SS -> SS S -> empty string	caresses -> caress ponies -> poni caress -> caress cats -> cat
a. (m > 0) AND Word ends with EED b. Word ends with ED or ING, and the normal form contains at least one vowel. If rule b is satisfied, then to clean the representation, also do one of the following. a. Word ends with AT or BL or IZ b. Word ends in 2 consonants AND the last letter isn’t L or S or Z c. (m=1) AND word ends in the form CVC in normalised form AND the second C of this CVC form isn’t W or X or Y	a. Replace EED -> EE b. Replace ED or ING by empty string. a. Add an E to the end of the word. b. Remove the last letter c. Add an E to the end of the word	feed -> feed agreed -> agree singing -> sing conflat (ed) -> conflate hopp (ing) -> hop fil (ing) -> file

If the word ends in Y, and the rest of the word contains a vowel.	Replace Y by I.	happy -> happi sky -> sky
2. Handle Common Suffixes		
If (m > 0) and word ends in the suffix on the left (see adjacent column), replace it by the item on the right.	ational -> ate tional -> tion enci -> ence anci -> ance izer -> ize abli -> able alli -> al entli -> ent eli -> e ousli -> ous ization -> ize ation -> ate ator -> ate alism -> al iveness -> ive fulness -> ful ousness- >ous aliti -> al iviti -> ive biliti -> ble icate -> ic ative -> empty string alize -> al iciti -> ic ful -> empty string ness -> empty string	relational -> relate conditional -> condition rational -> rational valenci -> valence hesitanci -> hesitance digitizer -> digitize conformabli -> conformab radicalli -> radical differentli -> different vileli - > vile analogousli -> analogous vietnamization -> vietnamiz predication -> predicate operator -> operate feudalism -> feudal decisiveness -> decisive hopefulness -> hopeful callousness -> callous formaliti -> formal sensitiviti -> sensitive sensibiliti -> sensible triplicate -> triplic formative -> form formalize -> formal electriciti -> electric electrical -> electric hopeful -> hope goodness -> good
If (m > 1) and word ends in the suffix (see adjacent column), replace it by an empty string	al ance ence er ic able	revival -> reviv allowance -> allow inference -> infer airliner -> airlin gyroscopic -> gyroscop adjustable -> adjust

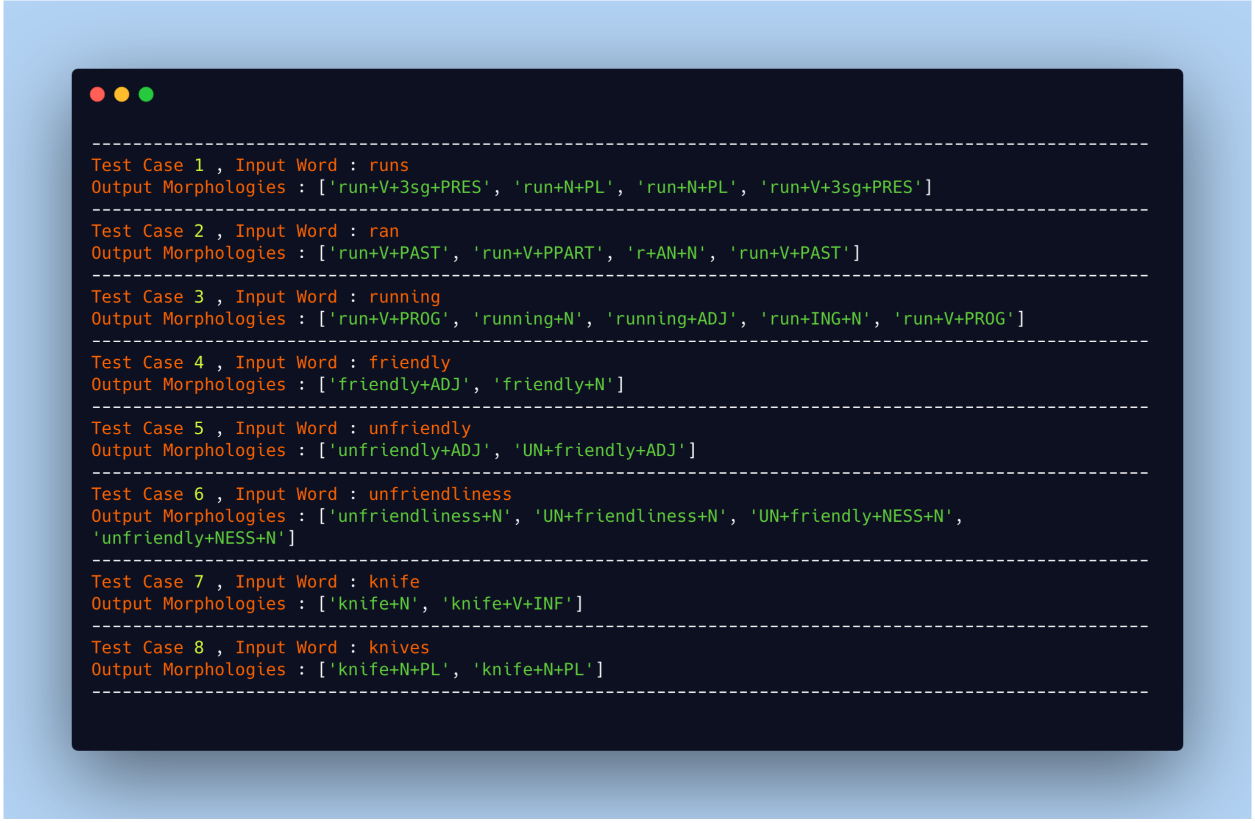
If the word ends in “**ion**”, and (m>1) and the last letter of the stemmed word is either S or T, then replace “**ion**” by an empty string.	ible ant ement ment ent ou ism ate iti ous ive ize	defensible -> defens irritant -> irrit replacement -> replac adjustment -> adjust dependent -> depend homologou -> homolog communism -> commun activate -> activ angulariti -> angular homologous -> homolog effective -> effect bowdlerize -> bowdler adopt***ion*** -> adopt
3. Handling words ending in “e” or “ll”		
If (m>1) OR ((m=1) AND word doesn’t end in a CVC form where the second C is a W or X or Y, replace the ending **“e”** by an empty string.		probate -> probat rate -> rate cease -> cease
If (m>1) and word ends in **“ll”**, then remove the last **“l”**		controll -> control roll -> rol

The algorithm doesn’t remove a suffix when the stem is too short, the length of the stem given by the measure **m** from the normalised form of the word.

The above algorithm is fairly accurate, and matches the output from NLTK for all the examples given here. Additionally, because the algorithm also removes prefixes before removing suffixes, it is sometimes more accurate for words with both prefixes and suffixes.

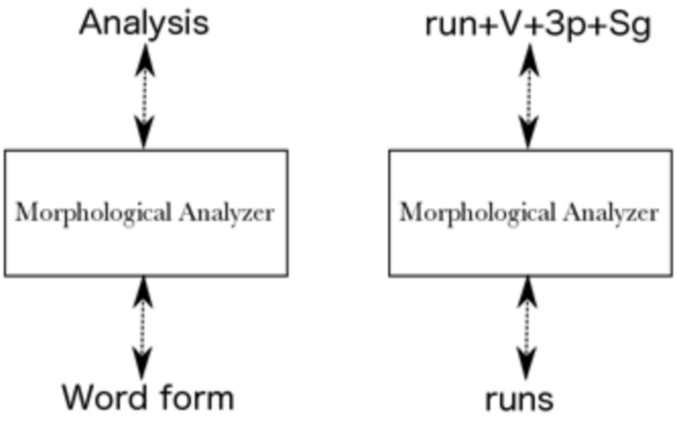
Morphological Analysis

The picture below shows the output from the morphological analysis performed for some of the words from the class slides.



```
-------------------------------------------------------------------------------------------------------
Test Case 1 , Input Word : runs
Output Morphologies : ['run+V+3sg+PRES', 'run+N+PL', 'run+N+PL', 'run+V+3sg+PRES']
-------------------------------------------------------------------------------------------------------
Test Case 2 , Input Word : ran
Output Morphologies : ['run+V+PAST', 'run+V+PPART', 'r+AN+N', 'run+V+PAST']
-------------------------------------------------------------------------------------------------------
Test Case 3 , Input Word : running
Output Morphologies : ['run+V+PROG', 'running+N', 'running+ADJ', 'run+ING+N', 'run+V+PROG']
-------------------------------------------------------------------------------------------------------
Test Case 4 , Input Word : friendly
Output Morphologies : ['friendly+ADJ', 'friendly+N']
-------------------------------------------------------------------------------------------------------
Test Case 5 , Input Word : unfriendly
Output Morphologies : ['unfriendly+ADJ', 'UN+friendly+ADJ']
-------------------------------------------------------------------------------------------------------
Test Case 6 , Input Word : unfriendliness
Output Morphologies : ['unfriendliness+N', 'UN+friendliness+N', 'UN+friendly+NESS+N',
'unfriendly+NESS+N']
-------------------------------------------------------------------------------------------------------
Test Case 7 , Input Word : knife
Output Morphologies : ['knife+N', 'knife+V+INF']
-------------------------------------------------------------------------------------------------------
Test Case 8 , Input Word : knives
Output Morphologies : ['knife+N+PL', 'knife+N+PL']
-------------------------------------------------------------------------------------------------------
```

The morphological analyzer I use can be thought of as a black box that functions as shown in the picture below.



The analyzer itself is a transducer written using the OpenFST backend for the library HFST .

A lexicon is defined for all the commonly known morphologies for the english language and the lexicon is compiled using hfst to produce a transducer with nearly 1.3 million transitions. This transducer is stored as a dictionary, with key = (source state, input symbol) and value = (destination state, output symbol). This format is also known as the ATT format for storing FSTs.

For the purpose of this assignment, we load this existing dictionary directly and perform lookups using it to get the most relevant morphologies of words. All the most probably morphologies are returned as output, as can be seen in the picture above.