Experiment No.4
Perform morphological analysis and word generation for any given text.
Date of Performance:
Date of Submission:

Aim: Perform morphological analysis and word generation for any given text.

Objective: Understand the working of stemming algorithms and apply stemming on the given input text.

Theory:

Stemming is a process of linguistic normalization, which reduces words to their word root word or chops off the derivational affixes. For example, connection, connected, connecting word reduce to a common word "conect".

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words "chocolates", "chocolatey", "choco" to the root word, "chocolate" and "retrieval", "retrieved", "retrieves" and reduces to the stem "retrieve". Stemming is an important part of the pipelining process in Natural language processing. The input to the stemmer is tokenized words.

Applications of stemming :

1. Stemming is used in information retrieval systems like search engines.
2. It is used to determine domain vocabularies in domain analysis.

Porter's Stemmer Algorithm:

It is one of the most popular stemming methods proposed in 1980. It is based on the idea that the suffixes in the English language are made up of a combination of smaller and simpler suffixes. This stemmer is known for its speed and simplicity. The main applications of Porter Stemmer include data mining and Information retrieval. However, its applications are only limited to English words. Also, the group of stems is mapped on to the same stem and the output stem is not necessarily a meaningful word. The algorithms are fairly lengthy in nature and are known to be the oldest stemmer.

Example: EED -> EE means "if the word has at least one vowel and consonant plus EED ending, change the ending to EE" as 'agreed' becomes 'agree'.

Advantage: It produces the best output as compared to other stemmers and it has less error rate.

Limitation: Morphological variants produced are not always real words.

Implementation:

```
import nltk

from nltk.tokenize import word_tokenize

from nltk.corpus import wordnet

from nltk.stem import WordNetLemmatizer

from nltk.tag import pos_tag



nltk.download('punkt')

nltk.download('averaged_perceptron_tagger')

nltk.download('wordnet')



def morphological_analysis(text):

    # Tokenize the text

    tokens = word_tokenize(text)



    # Part-of-speech tagging

    pos_tags = pos_tag(tokens)
```

```
    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    lemmas = [(word, lemmatizer.lemmatize(word, pos=get_wordnet_pos(tag))) for word, tag in pos_tags]

    return pos_tags, lemmas

def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN
```

```
def generate_words(word):
    synonyms = []

    for syn in wordnet.synsets(word):
        for lemma in syn.lemmas():
            synonyms.append(lemma.name())

    return set(synonyms)

text = "The quick brown fox jumps over the lazy dog."

pos_tags, lemmas = morphological_analysis(text)

word = "quick"
synonyms = generate_words(word)
print("Part-of-Speech Tags:", pos_tags)
print("Lemmas:", lemmas)
print(f"Synonyms for '{word}':", synonyms)
```

```
Part-of-Speech Tags: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jump
s', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), ('.', '.')]
Lemmas: [('The', 'The'), ('quick', 'quick'), ('brown', 'brown'), ('fox', 'fox'), ('jumps', 'j
ump'), ('over', 'over'), ('the', 'the'), ('lazy', 'lazy'), ('dog', 'dog'), ('.', '.')]
Synonyms for 'quick': {'spry', 'quickly', 'prompt', 'speedy', 'straightaway', 'flying', 'imme
diate', 'warm', 'nimble', 'quick', 'ready', 'fast', 'promptly', 'agile'}
```

Conclusion:

Comment on the implementation of stemming for an Indian language. Comment on the implementation of stemming for English (Explain which rules have been applied for identifying the stem words in your output).

Stemming in Indian Languages:
Indian languages, like Hindi or Tamil, have complex morphology with many inflections (gender, tense, case). Stemming needs to account for rich affixes (prefixes/suffixes) and compounding of words. A rule-based approach might not be enough, so statistical methods or hybrid approaches work better.

Stemming in English:
Your code uses **lemmatization**, not stemming. Stemming just strips affixes (e.g., `-ing`, `-ed`), while lemmatization considers the word's part of speech (POS) to find the base form.

In your output:

- POS Tagging: Tags each word's part of speech.
- Lemmatization: Uses tags to find base forms, like "jumps" → "jump".

Stemming is faster but less accurate, while lemmatization is context-aware and produces correct base forms.