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.

Code:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tag import pos_tag
# Download necessary NLTK resources (uncomment if you haven't downloaded them yet)
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
### Function for Morphological Analysis
def morphological_analysis(text):
    # Tokenize the text into words
    tokens = word_tokenize(text)
        # Part-of-speech tagging
    pos_tags = pos_tag(tokens)
        # Lemmatization
    lemmatizer = WordNetLemmatizer()
        # Creating a list of (word, lemma) pairs based on POS tagging
    lemmas = [(word, lemmatizer.lemmatize(word, pos=get_wordnet_pos(tag))) for word, tag
in pos_tags]
        return pos_tags, lemmas
### Function to map NLTK POS tags to WordNet POS tags
def get_wordnet_pos(tag):
    """Converts NLTK POS tags to WordNet POS tags."""
    if tag.startswith('J'):
        return wordnet.ADJ  # Adjective
    elif tag.startswith('V'):
        return wordnet.VERB  # Verb
    elif tag.startswith('N'):
        return wordnet.NOUN  # Noun
    elif tag.startswith('R'):
        return wordnet.ADV  # Adverb
    else:
        return wordnet.NOUN  # Default to noun if no tag matches
### Function for word generation (synonyms)
def generate_words(word):
    """Generates a set of synonyms for a given word."""
    synonyms = set()  # Using a set to avoid duplicate synonyms
        for syn in wordnet.synsets(word):
        for lemma in syn.lemmas():
            synonyms.add(lemma.name())  # Add synonym to the set
    return synonyms
### Example text
text = "The quick brown fox jumps over the lazy dog."
### Perform morphological analysis
pos_tags, lemmas = morphological_analysis(text)
### Generate words for a specific example
```

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

Output:

```
(venv) PS D:\Vartak college\sem 7\NLP\EXP\New folder> python .\exp4.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Mokshad\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Mokshad\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Mokshad\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
Part-of-Speech Tags: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), ('.', '.')]
Lemmas: [('The', 'The'), ('quick', 'quick'), ('brown', 'brown'), ('fox', 'fox'), ('jumps', 'jump'), ('over', 'over'), ('the', 'the'), ('lazy', 'lazy'), ('dog', 'dog'), ('.', '.')]
Synonyms for 'quick': {'ready', 'immediate', 'prompt', 'warm', 'fast', 'spry', 'nimble', 'quick', 'quickly', 'flying', 'straightaway', 'promptly', 'speedy', 'agile'}
```

Conclusion:

Implementation of Stemming for Indian Languages Stemming in Indian languages can be complex due to rich morphological structures, including inflections and derivations. Here's how stemming can be implemented:

1. Tools: Libraries like the Indic NLP Library provide specific stemming algorithms for various Indian languages (e.g., Hindi, Tamil). Custom algorithms may also be developed to handle language-specific morphological rules.
2. Approach:
   - Tokenization: Break down the text into individual words.
   - Suffix Removal: Identify and remove common suffixes based on linguistic rules.
3. Root Extraction: Reduce words to their base or root form
4. Challenges: The high morphological richness and variations among Indian languages necessitate tailored stemming approaches, often requiring substantial linguistic knowledge.

Implementation of Stemming for English: Stemming in English is often performed using established algorithms like the Porter Stemmer, which follows specific rules to reduce words to their base forms: a. Common Rules: Suffix Removal b. Step-wise Reduction: The process is iterative, applying multiple steps to refine the stem further until no more rules can be applied (e.g., "consigning" → "consign").