Python | Stemming words with NLTK

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”, and “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”.

> Prerequisite: Introduction to Stemming

```
Some more example of stemming for root word "like" include:

-> "likes"
-> "liked"
-> "likely"
-> "liking"
```

Errors in Stemming: There are mainly two errors in stemming – ***Overstemming*** and ***Understemming***. Overstemming occurs when two words are stemmed from the same root that are of different stems. Under-stemming occurs when two words are stemmed from the same root that is not of different stems.

Applications of stemming are:

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

Stemming is desirable as it may reduce redundancy as most of the time the word stem and their inflected/derived words mean the same.

Below is the implementation of stemming words using NLTK:

Code #1:

Python3

```
from  nltk.stem  import  PorterStemmer

from  nltk.tokenize  import  word_tokenize

ps  =  PorterStemmer()

words  =  [``"program"``,  "programs"``,  "programmer"``,  "programming"``,  "programmers"``]
```

```
for  w  in  words:

  print``(w,  " : "``, ps.stem(w))
```

Output:

```
program  :  program
programs  :  program
programmer  :  program
programming  :  program
programmers  :  program
```

Code #2: Stemming words from sentences

Python3

```
from  nltk.stem  import  PorterStemmer

from  nltk.tokenize  import  word_tokenize

ps  =  PorterStemmer()

sentence  =  "Programmers program with programming languages"

words  =  word_tokenize(sentence)

for  w  in  words:

  print``(w,  " : "``, ps.stem(w))
```

**Output 😗*

```
Programmers  :  program
program  :  program
with  :  with
programming  :  program
languages  :  language
```

Code #3: Using reduce():

**Algorithm 😗*

1. Import the necessary modules: PorterStemmer and word_tokenize from nltk, and reduce from functools.

2. Create an instance of the PorterStemmer class.
3. Define a sample sentence to be stemmed.
4. Tokenize the sentence into individual words using word_tokenize.
5. Use reduce to apply the PorterStemmer to each word in the tokenized sentence, and join the stemmed words back into a string.
6. Print the stemmed sentence.

```
install the pip install nltk
```

Python3

```
from  nltk.stem  import  PorterStemmer

from  nltk.tokenize  import  word_tokenize

from  functools  import  reduce

ps  =  PorterStemmer()

sentence  =  "Programmers program with programming languages"

words  =  word_tokenize(sentence)

stemmed_sentence  =  reduce``(``lambda  x, y: x  +  " "  +  ps.stem(y), words, "")

print``(stemmed_sentence)
```

Output:

Programm program with program language

Time complexity:
The time complexity of this code is O(nlogn), where n is the length of the input sentence. The tokenizer and stemmer functions have a linear time complexity of O(n), but the reduce function has a logarithmic time complexity of O(logn) since it processes elements in pairs.

Space complexity:
The space complexity of this code is O(n), where n is the length of the input sentence. This is because the reduce function creates a new string object that has the same length as the input sentence. The tokenizer and stemmer functions do not increase the space complexity significantly.