


```

    if (count==1):
        y.append(pair)
    return(y)

```

```

bigram = removal(bigram)
trigram = removal(trigram)
fourgram = removal(fourgram)
freq_bi = nltk.FreqDist(bigram)
freq_tri = nltk.FreqDist(trigram)
freq_four = nltk.FreqDist(fourgram)
print("Most common n-grams without stopwords removal and without add-1 smoothing: \n")
print ("Most common bigrams: ", freq_bi.most_common(5))
print ("\nMost common trigrams: ", freq_tri.most_common(5))
print ("\nMost common fourgrams: ", freq_four.most_common(5))

```

Most common n-grams without stopwords removal and without add-1 smoothing:

```

Most common bigrams:  [('said', 'the'), 209], [('said', 'alice'), 115], [('the', 'queen'), 65], [('the', 'king'), 60], [('a', 'lit'
Most common trigrams:  [('the', 'mock', 'turtle'), 51], [('the', 'march', 'hare'), 30], [('said', 'the', 'king'), 29], [('the', 'w
Most common fourgrams:  [('said', 'the', 'mock', 'turtle'), 19], [('she', 'said', 'to', 'herself'), 16], [('a', 'minute', 'or', 't

```

▼ Script for downloading the stopwords using NLTK

```

from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

```

▼ Print 10 Unigrams and Bigrams after removing stopwords

```

print("Most common n-grams with stopwords removal and without add-1 smoothing: \n")
unigram_sw_removed = [p for p in unigram if p not in stop_words]
fdist = nltk.FreqDist(unigram_sw_removed)
print("Most common unigrams: ", fdist.most_common(10))
bigram_sw_removed = []
bigram_sw_removed.extend(list(ngrams(unigram_sw_removed, 2)))
fdist = nltk.FreqDist(bigram_sw_removed)
print("\nMost common bigrams: ", fdist.most_common(10))

```

Most common n-grams with stopwords removal and without add-1 smoothing:

```

Most common unigrams:  ['said', 462], ['alice', 385], ['little', 128], ['one', 101], ['like', 85], ['know', 85], ['would', 83], ('
Most common bigrams:  [('said', 'alice'), 122], [('mock', 'turtle'), 54], [('march', 'hare'), 31], [('said', 'king'), 29], (('thou

```

▼ Add-1 smoothing

```

ngrams_all = {1:[], 2:[], 3:[], 4:[]}
for i in range(4):
    for each in tokenized_text:
        for j in ngrams(each, i+1):
            ngrams_all[i+1].append(j)
ngrams_voc = {1:set([]), 2:set([]), 3:set([]), 4:set([])}
for i in range(4):
    for gram in ngrams_all[i+1]:
        if gram not in ngrams_voc[i+1]:
            ngrams_voc[i+1].add(gram)
total_ngrams = {1:-1, 2:-1, 3:-1, 4:-1}
total_voc = {1:-1, 2:-1, 3:-1, 4:-1}
for i in range(4):
    total_ngrams[i+1] = len(ngrams_all[i+1])
    total_voc[i+1] = len(ngrams_voc[i+1])

ngrams_prob = {1:[], 2:[], 3:[], 4:[]}
for i in range(4):
    for ngram in ngrams_voc[i+1]:
        tlist = [ngram]
        tlist.append(ngrams_all[i+1].count(ngram))
        ngrams_prob[i+1].append(tlist)

for i in range(4):
    for ngram in ngrams_prob[i+1]:
        ngram[-1] = (ngram[-1]+1)/(total_ngrams[i+1]+total_voc[i+1])

```

Prints top 10 unigram, bigram, trigram, fourgram after smoothing

```
print("Most common n-grams without stopword removal and with add-1 smoothing: \n")
for i in range(4):
    ngrams_prob[i+1] = sorted(ngrams_prob[i+1], key = lambda x:x[1], reverse = True)

print ("Most common unigrams: ", str(ngrams_prob[1][:10]))
print ("\nMost common bigrams: ", str(ngrams_prob[2][:10]))
print ("\nMost common trigrams: ", str(ngrams_prob[3][:10]))
print ("\nMost common fourgrams: ", str(ngrams_prob[4][:10]))

Most common n-grams without stopword removal and with add-1 smoothing:

Most common unigrams: [[('the',), 0.05598462224968249], [('and',), 0.02900490852298081], [('to',), 0.02478289225277177], [('a',), 0.0229478289225277177], [('is',), 0.0229478289225277177], [('of',), 0.0229478289225277177], [('it',), 0.0229478289225277177], [('in',), 0.0229478289225277177], [('on',), 0.0229478289225277177], [('a',), 0.0229478289225277177]]

Most common bigrams: [[('said', 'the'), 0.0053395713087035016], [('of', 'the'), 0.0033308754354293268], [('said', 'alice'), 0.0029478289225277177], [('the', 'mock'), 0.0029478289225277177], [('the', 'march'), 0.0029478289225277177], [('the', 'hare'), 0.0029478289225277177], [('the', 'she'), 0.0029478289225277177], [('the', 'was'), 0.0029478289225277177], [('the', 'that'), 0.0029478289225277177], [('the', 'herself'), 0.0029478289225277177]]

Most common trigrams: [[('the', 'mock', 'turtle'), 0.001143837575064341], [('the', 'march', 'hare'), 0.0006819031697498955], [('sa', 'id', 'alice'), 0.0006819031697498955], [('the', 'she', 'was'), 0.0006819031697498955], [('the', 'that', 'herself'), 0.0006819031697498955], [('the', 'was', 'that'), 0.0006819031697498955], [('the', 'was', 'herself'), 0.0006819031697498955], [('the', 'was', 'that'), 0.0006819031697498955], [('the', 'was', 'herself'), 0.0006819031697498955], [('the', 'was', 'that'), 0.0006819031697498955]]

Most common fourgrams: [[('said', 'the', 'mock', 'turtle'), 0.00043521782652217433], [('she', 'said', 'to', 'herself'), 0.000369931697498955], [('the', 'said', 'to', 'herself'), 0.000369931697498955], [('the', 'said', 'to', 'herself'), 0.000369931697498955], [('the', 'said', 'to', 'herself'), 0.000369931697498955], [('the', 'said', 'to', 'herself'), 0.000369931697498955], [('the', 'said', 'to', 'herself'), 0.000369931697498955], [('the', 'said', 'to', 'herself'), 0.000369931697498955], [('the', 'said', 'to', 'herself'), 0.000369931697498955], [('the', 'said', 'to', 'herself'), 0.000369931697498955]]
```

Next word Prediction

```
str1 = 'after that alice said the'
str2 = 'alice felt so desperate that she was'

token_1 = word_tokenize(str1)
token_2 = word_tokenize(str2)
ngram_1 = {1:[], 2:[], 3:[]} #to store the n-grams formed
ngram_2 = {1:[], 2:[], 3:[]}
for i in range(3):
    ngram_1[i+1] = list(ngrams(token_1, i+1))[-1]
    ngram_2[i+1] = list(ngrams(token_2, i+1))[-1]
print("String 1: ", ngram_1, "\nString 2: ", ngram_2)

String 1: {1: ('the',), 2: ('said', 'the'), 3: ('alice', 'said', 'the')}
String 2: {1: ('was',), 2: ('she', 'was'), 3: ('that', 'she', 'was')}

for i in range(4):
    ngrams_prob[i+1] = sorted(ngrams_prob[i+1], key = lambda x:x[1], reverse = True)

pred_1 = {1:[], 2:[], 3:[]}
for i in range(3):
    count = 0
    for each in ngrams_prob[i+2]:
        if each[0][-1] == ngram_1[i+1]:
            count +=1
    #to find predictions based on highest probability of n-grams
    pred_1[i+1].append(each[0][-1])
    if count ==5:
        break
    if count<5:
        while(count!=5):
            pred_1[i+1].append("NOT FOUND")
            count +=1
#if no word prediction is found, replace with NOT FOUND
for i in range(4):
    ngrams_prob[i+1] = sorted(ngrams_prob[i+1], key = lambda x:x[1], reverse = True)

pred_2 = {1:[], 2:[], 3:[]}
for i in range(3):
    count = 0
    for each in ngrams_prob[i+2]:
        if each[0][-1] == ngram_2[i+1]:
            count +=1
    pred_2[i+1].append(each[0][-1])
    if count ==5:
        break
    if count<5:
        while(count!=5):
            pred_2[i+1].append("\0")
            count +=1

print("Next word predictions for the strings using the probability models of bigrams, trigrams, and fourgrams\n")
print("String 1 - after that alice said the-\n")
```

```
print("Bigram model predictions: {}\\nTrigram model predictions: {}\\nFourgram model predictions: {}\\n" .format(pred_1[1], pred_1[2], pred_1[3]))
print("String 2 - alice felt so desperate that she was-\\n")
print("Bigram model predictions: {}\\nTrigram model predictions: {}\\nFourgram model predictions: {}" .format(pred_2[1], pred_2[2], pred_2[3]))
```

Next word predictions for the strings using the probability models of bigrams, trigrams, and fourgrams

String 1 - after that alice said the-

```
Bigram model predictions: ['queen', 'king', 'gryphon', 'mock', 'hatter']
Trigram model predictions: ['king', 'hatter', 'mock', 'caterpillar', 'gryphon']
Fourgram model predictions: ['NOT FOUND', 'NOT FOUND', 'NOT FOUND', 'NOT FOUND', 'NOT FOUND']
```

String 2 - alice felt so desperate that she was-

```
Bigram model predictions: ['a', 'the', 'not', 'that', 'going']
Trigram model predictions: ['now', 'quite', 'a', 'beginning', 'walking']
Fourgram model predictions: ['now', 'ready', 'quite', 'dozing', 'in']
```