Connected to miniconda3 (Python 3.12.2)

In [ ]:
```python
import nltk
import random


def generate_ngram(n, corpus):
    # ngram is a dictionary that stores the list of words and the count of the next_word occuring after certain list of words.
    ngram = {}
    for i in range(len(corpus) - n + 1):
        # words_list framed as a tuple, as it should be immutable
        words_list = tuple(corpus[i : i + n - 1])
        next_word = corpus[i + n - 1]
        if words_list not in ngram:
            ngram[words_list] = {}
        if next_word not in ngram[words_list]:
            ngram[words_list][next_word] = 0
        # if words_list and next_word are both in the dictionary, then increase the count
        ngram[words_list][next_word] += 1
    return ngram


def predict_next_word(sentence, n, corpus, randomize=False, alpha=0.4):
    ngram = generate_ngram(n, corpus)
    highest_word = None

    # n should not be larger than length of sentence. The judgement between n and length of sentence is done in the finish_sentence function.
    context = tuple(sentence[-(n - 1) :])
    # first determine if context exists in the ngram, and if there is any next_word existing for the context. If so, no need to trigger backoff
    if context in ngram and ngram[context]:
        possible_words = ngram[context]
        # possible_words is a dictionary, key is the next_word, and value is the count of next_word
        total_count = sum(possible_words.values())
        # create another new dictionary that summarizes probability of the next_word probability
        possible_words_probs = {}
        for word, num in possible_words.items():
            prob = num / total_count
            possible_words_probs[word] = prob
            # when the probability is equal, choose the word that is first alphabetically
            if randomize == False:
                highest_word = max(
                    possible_words_probs,
                    key=lambda w: (possible_words_probs[w], -ord(w[0])),
                )
            # when randomize = True, we select randomly from the distribution, with weights being its respective probability
            else:
                highest_word = random.choices(
                    list(possible_words_probs.keys()),
                    weights=possible_words_probs.values(),
                )[0]
    # if context did not exist in the ngram, trigger backoff, calculate stupid backoff all the way back to n=1 with the penalty alpha powered accordingly
    else:
        possible_words_probs = {}
        for i in range(n - 1, 0, -1):
            context = tuple(sentence[-(i - 1) :])
            new_gram = generate_ngram(i, corpus)
            if context in new_gram and new_gram[context]:
                possible_words = new_gram[context]
                total_count = sum(possible_words.values())
                for word, num in possible_words.items():
                    # if the words already appeared in the dictionary, no need to calculate the probability again
                    if word not in possible_words_probs:
                        prob = (num / total_count) * (alpha ** (n - i))
                        possible_words_probs[word] = prob
        # after iterating through all the (n-1)gram to unigram, find the highest probability when randomize = false
        if randomize == False:
            highest_word = max(
                possible_words_probs,
                key=lambda w: (possible_words_probs[w], -ord(w[0])),
            )
        else:
            highest_word = random.choices(
                list(possible_words_probs.keys()), weights=possible_words_probs.values()
            )[0]

    return highest_word


def finish_sentence(sentence, n, corpus, randomize=False, alpha=0.4):
    "returns an extended sentence until"
    "the first ., ?, or ! is found OR until it has 10 total tokens"
    stop_tokens = {".", "?", "!"}
    # If n is larger than (length of sentence + 1), then we need to first cut n to the (length of sentence + 1). Use the whole sentence as context to predict next word.
    if n > (len(sentence) + 1):
        temp_n = len(sentence) + 1
        while len(sentence) < n - 1:
            next_word = predict_next_word(sentence, temp_n, corpus, randomize, alpha)
            if not next_word:
                break
            sentence.append(next_word)
            if next_word in stop_tokens:
                break

    while len(sentence) < 10:
        next_word = predict_next_word(sentence, n, corpus, randomize, alpha)
        if not next_word:
            break
        sentence.append(next_word)
        if next_word in stop_tokens:
            break

    return sentence
```

In [ ]:
```python
print("-------------Testing--------------")
corpus = tuple(
    nltk.word_tokenize(nltk.corpus.gutenberg.raw("austen-sense.txt").lower())
)
print("-------------Application #1 (Deterministic) --------------")
n = 4
st = ["he", "did"]
print("Input:{}, N={}".format(" ".join(st), n))
output = finish_sentence(st, n, corpus, randomize=False, alpha=0.4)
print("Output:{}".format(" ".join(output)))

print("-------------Application #2 (Deterministic) --------------")
n = 3
st = ["he", "did"]
print("Input:{}, N={}".format(" ".join(st), n))
output = finish_sentence(st, n, corpus, randomize=False, alpha=0.4)
print("Output:{}".format(" ".join(output)))

print("-------------Application #3 (Stochastic) --------------")
n = 3
st = ["he", "did"]
print("Input:{}, N={}".format(" ".join(st), n))
output = finish_sentence(st, n, corpus, randomize=True, alpha=0.4)
print("Output:{}".format(" ".join(output)))

print("-------------Application #4 (Deterministic) --------------")
n = 3
st = ["it", "was"]
print("Input:{}, N={}".format(" ".join(st), n))
output = finish_sentence(st, n, corpus, randomize=False, alpha=0.4)
print("Output:{}".format(" ".join(output)))

print("-------------Application #5 (Deterministic)--------------")
n = 4
st = ["it", "was"]
print("Input:{}, N={}".format(" ".join(st), n))
output = finish_sentence(st, n, corpus, randomize=False, alpha=0.4)
print("Output:{}".format(" ".join(output)))

print("-------------Application #6 (Stochastic)--------------")
n = 4
st = ["it", "was"]
print("Input:{}, N={}".format(" ".join(st), n))
output = finish_sentence(st, n, corpus, randomize=True, alpha=0.4)
print("Output:{}".format(" ".join(output)))

print("-------------Application #7 (Deterministic)--------------")
n = 4
st = ["they", "would"]
print("Input:{}, N={}".format(" ".join(st), n))
output = finish_sentence(st, n, corpus, randomize=False, alpha=0.4)
print("Output:{}".format(" ".join(output)))

print("-------------Application #8 (Deterministic)--------------")
n = 3
st = ["they", "would"]
print("Input:{}, N={}".format(" ".join(st), n))
output = finish_sentence(st, n, corpus, randomize=False, alpha=0.4)
print("Output:{}".format(" ".join(output)))

print("-------------Application #9 (Stochastic)--------------")
n = 3
st = ["they", "would"]
print("Input:{}, N={}".format(" ".join(st), n))
output = finish_sentence(st, n, corpus, randomize=True, alpha=0.4)
print("Output:{}".format(" ".join(output)))

print("-------------Application #10 (Deterministic)--------------")
n = 5
st = ["they", "would", "ask", "him"]
print("Input:{}, N={}".format(" ".join(st), n))
output = finish_sentence(st, n, corpus, randomize=False, alpha=0.4)
print("Output:{}".format(" ".join(output)))

print("-------------Application #11 (Deterministic)--------------")
n = 3
st = ["they", "would", "ask", "him"]
print("Input:{}, N={}".format(" ".join(st), n))
output = finish_sentence(st, n, corpus, randomize=False, alpha=0.4)
print("Output:{}".format(" ".join(output)))

print("-------------Application #12 (Stochastic)--------------")
n = 3
st = ["they", "would", "ask", "him"]
print("Input:{}, N={}".format(" ".join(st), n))
output = finish_sentence(st, n, corpus, randomize=True, alpha=0.4)
print("Output:{}".format(" ".join(output)))
```

```
-------------Testing--------------
-------------Application #1 (Deterministic) --------------
Input:he did, N=4
Output:he did not feel the continuance of his existence secure
-------------Application #2 (Deterministic) --------------
Input:he did, N=3
Output:he did not know what you are very much to
-------------Application #3 (Stochastic) --------------
Input:he did, N=3
Output:he did nothing but only civility ?
-------------Application #4 (Deterministic) --------------
Input:it was, N=3
Output:it was not in the world , and the two
-------------Application #5 (Deterministic)--------------
Input:it was, N=4
Output:it was not a thing to be thought of ;
-------------Application #6 (Stochastic)--------------
Input:it was, N=4
Output:it was a very great relief to you , in
-------------Application #7 (Deterministic)--------------
Input:they would, N=4
Output:they would all dine with lady middleton .
-------------Application #8 (Deterministic)--------------
Input:they would, N=3
Output:they would all be made for him , and the
-------------Application #9 (Stochastic)--------------
Input:they would, N=3
Output:they would soon , and then , '' cried sir
-------------Application #10 (Deterministic)--------------
Input:they would ask him, N=5
Output:they would ask him if there was any news .
-------------Application #11 (Deterministic)--------------
Input:they would ask him, N=3
Output:they would ask him if he had been in the
-------------Application #12 (Stochastic)--------------
Input:they would ask him, N=3
Output:they would ask him questions which his society bestowed on
```