N-Gram Model Implementation

In [1]:
```
import nltk
from nltk import ngrams
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
```

In [2]:
```
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:\Users\ASUS\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Out[2]: True

In [3]:
```
text = "How much wood would a woodchuck chuck could chuck wood,if a woodchuck could chuck wood"
```

In [4]:
```
tokens = word_tokenize(text.lower())
```

In [5]:
```
bigram_list = list(ngrams(tokens, 2))
trigram_list = list(ngrams(tokens, 3))
```

In [6]:
```
bigram_freq = FreqDist(bigram_list)
trigram_freq = FreqDist(trigram_list)
```

In [7]:
```
print("Bigrams:")
for bigram, frequency in bigram_freq.items():
    print(f"{bigram}: {frequency}")

print("\nTrigrams:")
for trigram, frequency in trigram_freq.items():
    print(f"{trigram}: {frequency}")
```

```
Bigrams:
('how', 'much'): 1
('much', 'wood'): 1
('wood', 'would'): 1
('would', 'a'): 1
('a', 'woodchuck'): 2
('woodchuck', 'chuck'): 1
('chuck', 'could'): 1
('could', 'chuck'): 2
('chuck', 'wood'): 2
('wood', ','): 1
(',', 'if'): 1
('if', 'a'): 1
('woodchuck', 'could'): 1

Trigrams:
('how', 'much', 'wood'): 1
('much', 'wood', 'would'): 1
('wood', 'would', 'a'): 1
('would', 'a', 'woodchuck'): 1
('a', 'woodchuck', 'chuck'): 1
('woodchuck', 'chuck', 'could'): 1
('chuck', 'could', 'chuck'): 1
('could', 'chuck', 'wood'): 2
('chuck', 'wood', ','): 1
('wood', ',', 'if'): 1
(',', 'if', 'a'): 1
('if', 'a', 'woodchuck'): 1
('a', 'woodchuck', 'could'): 1
('woodchuck', 'could', 'chuck'): 1
```

In [8]:
```
def predict_next_word(context_word, ngram_freq):
    candidates = {next_word: freq for (w1, next_word), freq in ngram_freq.items() if w1 == context_word}
    if not candidates:
        return None
    return max(candidates.items(), key=lambda x: x[1])[0]
```

In [9]:
```
context = 'woodchuck'
predicted_word = predict_next_word(context, bigram_freq)
print(f"\nPredicted word after '{context}': {predicted_word}")
```

```
Predicted word after 'woodchuck': chuck
```

In [10]:
```
# Tokenization and BoW
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
import nltk
nltk.download('punkt')

tokens = word_tokenize(text)
print("Tokens:", tokens)

vectorizer = CountVectorizer()
X = vectorizer.fit_transform([text])
```

```
print("Vocabulary:", vectorizer.get_feature_names_out())
print("BoW Matrix:", X.toarray())
```

```
Tokens: ['How', 'much', 'wood', 'would', 'a', 'woodchuck', 'chuck', 'could', 'chuck', 'wood', ',', 'if', 'a', 'woodchuck', 'could', 'chuck',
'wood']
Vocabulary: ['chuck' 'could' 'how' 'if' 'much' 'wood' 'woodchuck' 'would']
BoW Matrix: [[3 2 1 1 1 3 2 1]]
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ASUS\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```