EXPT NO: 1 A)	**TEXT GENERATION USING N-GRAM LANGUAGE MODEL**
DATE: 04/02/26	

AIM

Implement an N-Gram–based predictive text model using bigram and trigram probability distributions. This helps to understand how contextual probability influences next-word prediction.

ALGORITHM/ STEPS

- **STEP 1:** Initialize the environment by installing and importing the Natural Language Toolkit (nltk). Download the Brown Corpus (for text data) and the Universal Tagset (for simplified Part-of-Speech tags).
- **STEP 2:** Extract raw sentences from the Brown Corpus using brown.sents().
- **STEP 3: Tokenization:** Convert all words to lowercase and flatten the nested list into a single list of tokens to verify the total volume of training data.
- **STEP 4:** Build an N-Gram Model for Bigram and Trigram.
- **STEP 5:** Predict Next Word and display Top-5 Suggestions.
- **STEP 6:** Test Bigram & Trigram Prediction on certain words

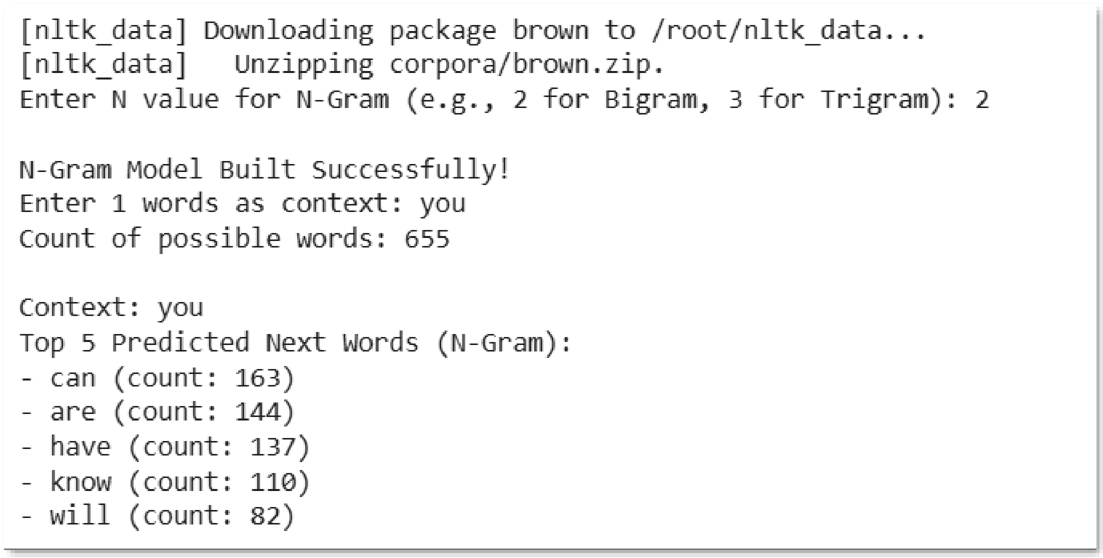
CODE IMPLEMENTATION

```
import nltk
from nltk.corpus import brown
from collections import defaultdict, Counter
nltk.download('brown')
```

```python
# Load and preprocess
sentences = brown.sents()
cleaned_sentences = []
for sent in sentences:
    cleaned = []
    for w in sent:
        if w.isalpha():
            cleaned.append(w.lower())
    cleaned_sentences.append(cleaned)

# -------- USER INPUT FOR N --------
n = int(input("Enter N value for N-Gram(e.g: 2 for Bigram, 3 for Trigram): 
"))

# Build N-Gram Model
ngram_model = defaultdict(Counter)
for sent in cleaned_sentences:
    for i in range(len(sent) - n + 1):
        context = tuple(sent[i:i+n-1])
        next_word = sent[i+n-1]
        ngram_model[context][next_word] += 1

# Print the values
print("\nN-Gram Model Built Successfully!")

# -------- USER INPUT FOR CONTEXT --------

context_input = input(f"Enter {n-1} words as context: ")
context_words = tuple(context_input.lower().split())

# Validate context length
if len(context_words) != n-1:
    print(f"Error: Please enter exactly {n-1} words!")

else:
    # Predict next word
    if context_words in ngram_model:
        print("Count of possible words:", len(ngram_model[context_words]))
        top_suggestions = ngram_model[context_words].most_common(5)
        print("\nContext:", context_input)
        print("Top 5 Predicted Next Words (N-Gram):")
        for word, count in top_suggestions:
            print(f"- {word} (count: {count})")
    else:
        print("\nContext:", context_input)
        print("No prediction available for this context")
```

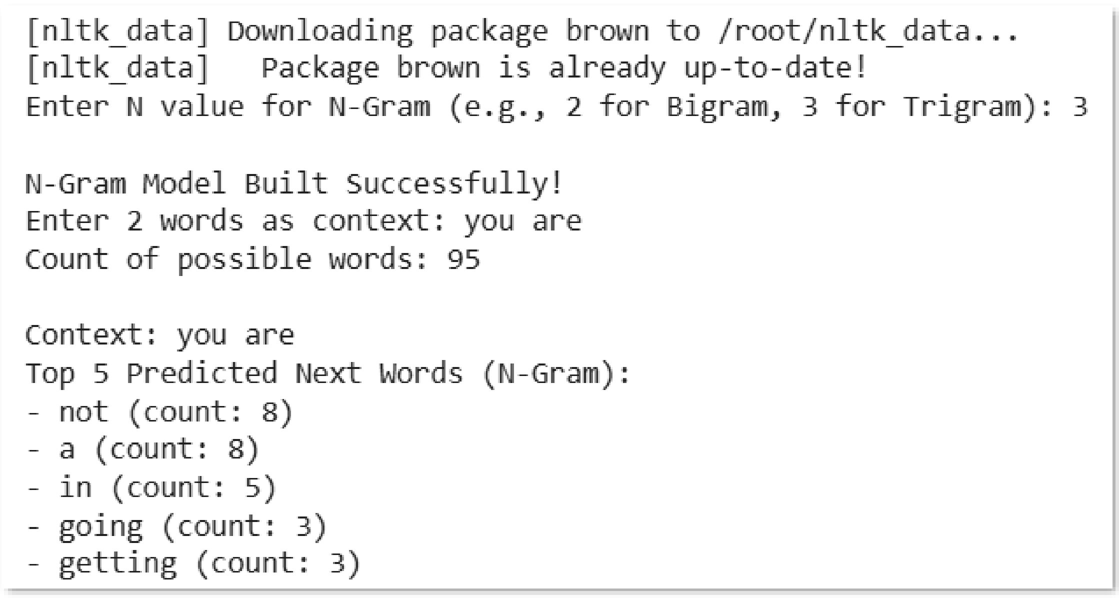
OUTPUT

BIGRAM MODEL



```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
Enter N value for N-Gram (e.g., 2 for Bigram, 3 for Trigram): 2

N-Gram Model Built Successfully!
Enter 1 words as context: you
Count of possible words: 655

Context: you
Top 5 Predicted Next Words (N-Gram):
- can (count: 163)
- are (count: 144)
- have (count: 137)
- know (count: 110)
- will (count: 82)
```

TRIGRAM MODEL



```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
Enter N value for N-Gram (e.g., 2 for Bigram, 3 for Trigram): 3

N-Gram Model Built Successfully!
Enter 2 words as context: you are
Count of possible words: 95

Context: you are
Top 5 Predicted Next Words (N-Gram):
- not (count: 8)
- a (count: 8)
- in (count: 5)
- going (count: 3)
- getting (count: 3)
```

RESULT

Thus, upon completion of this experiment, the bigram and trigram probability distributions for next-word prediction was implemented and evaluated using the Brown Corpus.