```
import math
import re
from collections import defaultdict, Counter

class NGramLanguageModel:
    def __init__(self, n):
        """
        Initialize the N-gram Language Model

        :param n: Order of the N-gram model (2 for bigram, 3 for trigram, etc.)
        """
        self.n = n
        self.vocabulary = set()
        self.unigram_counts = Counter()
        self.ngram_counts = defaultdict(Counter)
        self.total_tokens = 0
        self.smoothing_alpha = 0.1

    def preprocess_text(self, text):
        """
        Preprocess the input text

        :param text: Input text string
        :return: List of tokenized and cleaned words
        """
        # Convert to lowercase and split into words
        words = re.findall(r'\w+', text.lower())

        # Add start and end tokens
        padded_words = ['<s>'] * (self.n - 1) + words + ['</s>']

        return padded_words

    def train_unsmoothed(self, corpus):
        """
        Train the unsmoothed N-gram model

        :param corpus: List of text documents
        """
        # Reset model parameters
        self.unigram_counts = Counter()
        self.ngram_counts = defaultdict(Counter)
        self.vocabulary = set()
        self.total_tokens = 0

        # Process each document in the corpus
        for doc in corpus:
            tokens = self.preprocess_text(doc)

            # Update unigram counts
            self.unigram_counts.update(tokens)
            self.vocabulary.update(tokens)
            self.total_tokens += len(tokens)

            # Update N-gram counts
            for i in range(len(tokens) - self.n + 1):
```

```
            for i in range(len(tokens) - self.n + 1):
                context = tuple(tokens[i:i+self.n-1])
                current_word = tokens[i+self.n-1]
                self.ngram_counts[context][current_word] += 1

    def train_smoothed(self, corpus, alpha=0.1):
        """
        Train the smoothed (Laplace/Add-alpha) N-gram model

        :param corpus: List of text documents
        :param alpha: Smoothing parameter
        """
        # Reset model parameters
        self.unigram_counts = Counter()
        self.ngram_counts = defaultdict(Counter)
        self.vocabulary = set()
        self.total_tokens = 0
        self.smoothing_alpha = alpha

        # Process each document in the corpus
        for doc in corpus:
            tokens = self.preprocess_text(doc)

            # Update unigram counts
            self.unigram_counts.update(tokens)
            self.vocabulary.update(tokens)
            self.total_tokens += len(tokens)

            # Update N-gram counts
            for i in range(len(tokens) - self.n + 1):
                context = tuple(tokens[i:i+self.n-1])
                current_word = tokens[i+self.n-1]
                self.ngram_counts[context][current_word] += 1

    def calculate_probability_unsmoothed(self, context, word):
        """
        Calculate probability for unsmoothed model

        :param context: Preceding N-1 words
        :param word: Current word
        :return: Probability of the word given the context
        """
        context_count = sum(self.ngram_counts[context].values())
        if context_count == 0:
            return 0.0

        return self.ngram_counts[context][word] / context_count

    def calculate_probability_smoothed(self, context, word):
        """
        Calculate probability for smoothed (Laplace) model

        :param context: Preceding N-1 words
        :param word: Current word
        :return: Smoothed probability of the word given the context
        """
        # Number of unique words in vocabulary
        vocab_size = len(self.vocabulary)
```

```
        vocab_size = len(self.vocabulary)

        # Count of current n-gram
        ngram_count = self.ngram_counts[context][word]

        # Count of context
        context_count = sum(self.ngram_counts[context].values())

        # Laplace smoothing
        smoothed_prob = (ngram_count + self.smoothing_alpha) / \
                        (context_count + self.smoothing_alpha * vocab_size)

        return smoothed_prob

    def calculate_perplexity(self, test_corpus, smoothed=False):
        """
        Calculate perplexity of the model on test corpus

        :param test_corpus: List of text documents for testing
        :param smoothed: Whether to use smoothed probabilities
        :return: Perplexity score
        """
        log_prob_sum = 0
        total_test_tokens = 0

        for doc in test_corpus:
            tokens = self.preprocess_text(doc)

            for i in range(len(tokens) - self.n + 1):
                context = tuple(tokens[i:i+self.n-1])
                current_word = tokens[i+self.n-1]

                # Select probability calculation method
                if smoothed:
                    prob = self.calculate_probability_smoothed(context, current_word)
                else:
                    prob = self.calculate_probability_unsmoothed(context, current_word)

                # Avoid log(0)
                prob = max(prob, 1e-10)

                log_prob_sum += math.log(prob)
                total_test_tokens += 1

        # Calculate perplexity
        avg_log_prob = log_prob_sum / total_test_tokens
        perplexity = math.exp(-avg_log_prob)

        return perplexity

    def print_model_information(self):
        """
        Print comprehensive information about the trained N-gram model
        """
        print("\n--- N-gram Language Model Information ---")
        print(f"N-gram Order: {self.n}")
        print(f"Total Tokens: {self.total_tokens}")
        print(f"Vocabulary Size: {len(self.vocabulary)}")
```

```
        # Vocabulary Details
        print("\n--- Vocabulary ---")
        print("Top 10 Most Frequent Words:")
        for word, count in self.unigram_counts.most_common(10):
            print(f"{word}: {count}")

        # N-gram Details
        print(f"\n--- {self.n}-gram Statistics ---")
        print("Number of Unique Contexts:", len(self.ngram_counts))

        # Top N-grams
        print("\nTop 10 Most Frequent N-grams:")
        top_ngrams = []
        for context, word_counts in list(self.ngram_counts.items())[:10]:
            for word, count in word_counts.most_common(1):
                top_ngrams.append((context, word, count))

        for context, word, count in sorted(top_ngrams, key=lambda x: x[2], reverse=True):
            print(f"Context {context} -> Word '{word}': {count} times")

        # Probability Distribution
        print("\n--- Probability Distribution ---")
        print("Smoothing Alpha:", self.smoothing_alpha)

        # Sample Probability Calculations
        print("\nSample Probability Calculations:")
        contexts_to_sample = list(self.ngram_counts.keys())[:5]
        for context in contexts_to_sample:
            print(f"\nContext: {context}")
            # Get top 3 most probable words for this context
            top_words = sorted(
                [(word, self.calculate_probability_smoothed(context, word))
                 for word in self.ngram_counts[context].keys()],
                key=lambda x: x[1],
                reverse=True
            )[:3]

            for word, prob in top_words:
                print(f"  {word}: {prob:.4f}")
```

```
# Sample corpus
train_corpus = [
    "the quick brown fox jumps over the lazy dog",
    "a quick brown dog jumps over the lazy fox",
    "the lazy fox sleeps all day"
]

test_corpus = [
    "quick brown animal jumps",
    "lazy fox sleeps"
]
```

```
# Bigram model
bigram_model = NGramLanguageModel(n=2)
```

```
# Train unsmoothed model
bigram_model.train_unsmoothed(train_corpus)
unsmoothed_perplexity = bigram_model.calculate_perplexity(test_corpus, smoothed=False)
print(f"Unsmoothed Bigram Perplexity: {unsmoothed_perplexity}")
```

```
Unsmoothed Bigram Perplexity: 5485874.080739646
```

```
# Print detailed model information
bigram_model.print_model_information()
```

```
--- N-gram Language Model Information ---
N-gram Order: 2
Total Tokens: 30
Vocabulary Size: 14

--- Vocabulary ---
Top 10 Most Frequent Words:
the: 4
<s>: 3
fox: 3
lazy: 3
</s>: 3
quick: 2
brown: 2
jumps: 2
over: 2
dog: 2

--- 2-gram Statistics ---
Number of Unique Contexts: 14

Top 10 Most Frequent N-grams:
Context ('the',) -> Word 'lazy': 3 times
Context ('<s>',) -> Word 'the': 2 times
Context ('quick',) -> Word 'brown': 2 times
Context ('jumps',) -> Word 'over': 2 times
Context ('over',) -> Word 'the': 2 times
Context ('lazy',) -> Word 'fox': 2 times
Context ('brown',) -> Word 'fox': 1 times
Context ('fox',) -> Word 'jumps': 1 times
Context ('dog',) -> Word '</s>': 1 times
Context ('a',) -> Word 'quick': 1 times

--- Probability Distribution ---
Smoothing Alpha: 0.1

Sample Probability Calculations:

Context: ('<s>',)
  the: 0.4773
  a: 0.2500

Context: ('the',)
  lazy: 0.5741
  quick: 0.2037
```

```
Context: ('quick',)
  brown: 0.6176

Context: ('brown',)
  fox: 0.3235
  dog: 0.3235

Context: ('fox',)
  jumps: 0.2500
  </s>: 0.2500
```

```
# Train smoothed model
bigram_model.train_smoothed(train_corpus, alpha=0.1)
smoothed_perplexity = bigram_model.calculate_perplexity(test_corpus, smoothed=True)
print(f"\nSmoothed Bigram Perplexity: {smoothed_perplexity}")
```

```
Smoothed Bigram Perplexity: 12.94446006439085
```