```
import math

if __name__ == '__main__':
    training_corpus = []

    training = {}
    test_corpus = []
    test = {}

    #read in the training corpus, change case, pad the lines, and get frequencies
    with open("train-Spring2022.txt", "r", encoding="utf-8") as file:
        for line in file:
            line = line.lower()
            line = "<s> " + line + " </s>"
            line = line.split()
            training_corpus.append(line)
            for token in line:
                try:
                    training[token] += 1
                except:
                    training[token] = 1

    #read in the test data, change case, pad lines, and get frequencies
    with open("test.txt", "r", encoding="utf-8") as file:
        for line in file:
            line = line.lower()
            line = "<s> " + line + " </s>"
            line = line.split()
            test_corpus.append(line)
            for token in line:
                try:
                    test[token] += 1
                except:
                    test[token] = 1

    #take counts of the types and tokens in the test data, and count of those not in the
    # #training data
    total_test_tokens = 0
    total_test_types = len(test) - 1
    test_tokens_not_in_training = 0
    test_types_not_in_training = 0
    for key in test:
        total_test_tokens += test[key]
        if key not in training:
            test_tokens_not_in_training += test[key]
            test_types_not_in_training += 1
    total_test_tokens -= test["<s>"]


    print("Total Word Types in Test Data:", total_test_types)
    print("Total Word Types NOT in Training:", test_types_not_in_training)
    print("Percentage of Word Types Not in Training:",
            (test_types_not_in_training/total_test_types)*100, "%\n")
    print("Total Tokens in Test:", total_test_tokens)
    print("Total Tokens NOT in Training:", test_tokens_not_in_training)
    print("Percentage of Tokens Not in Training:",
            (test_tokens_not_in_training/total_test_tokens)*100, "%\n")

    training["<unk>"] = 0
    delList = []

    #replace singletons and record them for deletion
    for key in training:
        if training[key] == 1:
```

```
            training["<unk>"] += 1
            delList.append(key)

    #delete singletons
    for key in delList:
        del training[key]

    #replace singletons in training data for '<unk>'
    for line in training_corpus:
        for token in range(len(line)):
            if line[token] not in training:
                line[token] = "<unk>"

    #take count of all tokens the training corpus
    total_u = 0
    for key in training:
        total_u += training[key]
    total_u -= training["<s>"]
    print("There are", len(training)-1, "unique words in the training corpus. \n")
    print("There are", total_u, "total tokens in the training corpus\n")


    test["<unk>"] = 0
    delList = []

    #find which tokens in the test data are not in the training corpus
    for key in test:
        if key not in training:
            test["<unk>"] += test[key]
            delList.append(key)

    #delete all the tokens not seen in the training corpus
    for key in delList:
        del test[key]

    #replace the tokens not seen in the training corpus
    for line in test_corpus:
        for token in range(len(line)):
            if line[token] not in test:
                line[token] = "<unk>"

    #for each line in the training corpus, record the bigrams and their frequencies
    # bigram dictionaries use tuples are keys
    trainingB = {}
    testB = {}
    for line in training_corpus:
        for token in range(1, len(line)):
            try:
                trainingB[(line[token-1], line[token])] += 1
            except:
                trainingB[(line[token-1], line[token])] = 1

    #for each line in the test corpus, record the bigrams and their frequencies
    for line in test_corpus:
        for token in range(1, len(line)):
            try:
                testB[(line[token-1], line[token])] += 1
            except:
                testB[(line[token-1], line[token])] = 1

    #record the amount of bigram tokens and types in the test data, and see which are
    #not in the training data
    total_test_brigram_tokens = 0
    total_test_bigram_types = len(testB)
    test_birgram_tokens_not_in_training = 0
```

```
    test_bigram_types_not_in_training = 0
    for key in testB:
        total_test_brigram_tokens += testB[key]
        if key not in trainingB:
            test_bigram_types_not_in_training += 1
            test_birgram_tokens_not_in_training += testB[key]

    print("Total Bigram Types in Test Data:", total_test_bigram_types)
    print("Total Bigram Types NOT in Training:", test_bigram_types_not_in_training)
    print("Percentage of Bigram Types Not in Training:",
            (test_bigram_types_not_in_training/total_test_bigram_types)*100, "%")
    print("Total Bigram Tokens in Test:", total_test_brigram_tokens)
    print("Total Bigram Tokens NOT in Training:", test_birgram_tokens_not_in_training)
    print("Percentage of Bigram Tokens Not in Training:",
            (test_birgram_tokens_not_in_training/total_test_brigram_tokens)*100, "%", "\n")

    #enter sentence and preprocess
    s = "<s> I look forward to hearing your reply . </s>"
    s = s.lower()
    s = s.split()
    for token in range(len(s)):
        if s[token] not in training:
            s[token] = "<unk>"


    uMLE = 0
    bMLE = 0
    bAO = 0
    uMLETokens = []
    bMLETokens = []
    bAOTokens = []

    #compute the log weight probabilities for each model
    for token in range(1, len(s)):
        try:
            calc = math.log2(training[s[token]]/total_u)
            uMLE += calc
            uMLETokens.append((s[token], calc))
        except:
            uMLE += float('-inf')
            uMLETokens.append((s[token], '-inf'))
        try:
            calc = math.log2(trainingB[(s[token-1], s[token])]/training[s[token-1]])
            bMLE += calc
            bMLETokens.append((s[token-1] + " " + s[token], calc))
        except:
            bMLE += float('-inf')
            bMLETokens.append((s[token-1] + " " + s[token], '-inf'))
        try:
            calc = math.log2((trainingB[(s[token-1], s[token])] + 1)/(training[s[token-1]] +
 len(training)))
            bAO += calc
            bAOTokens.append((s[token-1] + " " + s[token], calc))
        except:
            if s[token-1] in training:
                calc = math.log2(1/(training[s[token-1]] + len(training)))
                bAO += calc
                bAOTokens.append((s[token-1] + " " + s[token], calc))
            else:
                bAO += float('-inf')
                bAOTokens.append((s[token-1] + " " + s[token], '-inf'))

    print("Unigram MLE Log Probability of Given Sentence:")
    for token in uMLETokens:
        print(token[0], ":", token[1])
```

```
    print("Total:", uMLE, "\n")
    print("Bigram MLE Log Probability of Given Sentence:")
    for token in bMLETokens:
        print(token[0], ":",token[1])
    print("Total:", bMLE, "\n")
    print("Bigram Add-One Smoothed Log Probability of Givn Sentence:")
    for token in bAOTokens:
        print(token[0], ":",token[1])
    print("Total:", bAO, "\n")


    # compute perplexities, unigram exludes <s>
    uMLEPerplexity = 0 - uMLE/(len(s) - 1)
    bMLEPerplexity = 0 - bMLE/(len(s))
    bAOPerplexity = 0 - bAO/(len(s))
    print("Unigram MLE Perplexity of Given Sentence:")
    print(2**uMLEPerplexity, "\n")
    print("Bigram MLE Perplexity of Given Sentence:")
    print(2**bMLEPerplexity, "\n")
    print("Bigram Add-One Smoothed Perplexity of Given Sentence:")
    print(2**bAOPerplexity, "\n")

    uMLE = 0
    bMLE = 0
    bAO = 0
    tokenCountU = 0
    tokenCountB = 0

    #compute the log weight probabilities for the test corpus under each model
    for line in test_corpus:
        tokenCountU += len(line)-1
        tokenCountB += len(line)
        for token in range(1, len(line)):
            try:
                uMLE += math.log2(training[line[token]]/total_u)
            except:
                uMLE += float('-inf')
            try:
                bMLE += math.log2(trainingB[(line[token-1], line[token])]/training[line[token-1]])
            except:
                bMLE += float('-inf')
            try:
                bAO += math.log2((trainingB[(line[token-1], line[token])] + 1)/(training[line[token-
1]] + len(training)))
            except:
                if line[token-1] in training:
                    bAO += math.log2(1/(training[line[token-1]] + len(training)))
                else:
                    bAO += float('-inf')

    #compute the perplexities for each model, unigrams ignore <s>
    uMLEPerplexity = -(uMLE/(tokenCountU -1))
    bMLEPerplexity = -(bMLE/tokenCountB)
    bAOPerplexity = -(bAO/tokenCountB)

    print("Unigram MLE Perplexity of Test Data:")
    print(2**uMLEPerplexity, "\n")
    print("Bigram MLE Perplexity of Test Data:")
    print(2**bMLEPerplexity, "\n")
    print("Bigram Add-One Smoothed Perplexity of Test Data:")
    print(2**bAOPerplexity, "\n")
```