```
# Importing

import numpy as np
import re


# Reading the Book and Preprocessing

# Reading the file
book = 'war_and_peace.txt'
file = open(book, 'r', encoding='utf-8')
text = file.read()
file.close()

# Removing the punctuations and converting to lower case
text = re.sub(r'[^a-zA-Z]', " ", text)
text = " ".join(text.split()).lower()[:100000]

# Creating a dictionary of all the unique characters
dictionary = {}
for i in range(26):
    dictionary[chr(i + 97)] = i
dictionary[" "] = 26
```

```
# Initialize the parameters

# Observed sequence
O = np.zeros(len(text), dtype=int)

for i in range(len(text)):
    O[i] = dictionary[text[i]]
# Initial state distribution
pi = np.array(([0.525483, 0.474517]))
# Observable sequence
B = np.array([[0.03735, 0.03408, 0.03455, 0.03828, 0.03782, 0.03922, 0.03688, 0.03408, 0.03875, 0.04062, 0.03735, 0.03968,
              [0.03909, 0.03537,  0.03537, 0.03909, 0.03583,  0.03630, 0.04048, 0.03537, 0.03816, 0.03909, 0.03490, 0.0372
# Transition matrix
A = np.array([[0.47468, 0.52532], [0.51656, 0.48344]])
# Set of possible observations
V = np.array(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v'
# Set of possible states, Q is hidden
# Number of observation symbols
M = len(V)
# Number of states in the model
N = len(A)
# Length of observation sequence
T = len(O)
```

```
# Alpha Pass
'''
The alpha pass is a forward algorithm that computes the probability of being in a particular state at a given time,
given the observed sequence up to that time. The forward variable alpha represents the joint probability of the observed
sequence up to a certain point and being in a particular state at that point. The final alpha value represents the total
probability of the observed sequence.
'''



def alpha_pass(A1, B1, pi1, O1):
    c1 = np.zeros([T, 1])
    alpha1 = np.zeros([T, N])
    c1[0][0] = 0
    for x in range(N):
        alpha1[0][x] = pi1[x] * B1[x][O1[0]]
        c1[0][0] = c1[0][0] + alpha1[0][x]
    c1[0][0] = 1/c1[0][0]
    for x in range(N):
        alpha1[0][x] = c1[0][0] * alpha1[0][x]

    for t in range(1, T):
        c1[t][0] = 0
        for x in range(N):
            alpha1[t][x] = 0
            for y in range(N):
                alpha1[t][x] = alpha1[t][x] + alpha1[t-1][y] * A1[y][x]
            alpha1[t][x] = alpha1[t][x] * B1[x][O1[t]]
            c1[t][0] = c1[t][0] + alpha1[t][x]
        c1[t][0] = 1/c1[t][0]
        for x in range(N):
            alpha1[t][x] = c1[t][0] * alpha1[t][x]
    return alpha1, c1
```

Beta Pass

'''
The beta pass is a backward algorithm that computes the probability of observing the remainder of the sequence from a given state at a given time. The beta pass involves recursively computing the backward variable beta, which represents the probability of observing the remainder of the sequence starting from a particular state at a particular time. The final beta value represents the probability of observing the entire sequence starting from a particular state at a particular time.
'''

```
def beta_pass(A1, B1, O1, c1):
    beta1 = np.zeros([T, N])
    for x in range(N):
        beta1[T-1][x] = c1[T-1][0]
    for t in range(T-2, -1, -1):
        for x in range(N):
            beta1[t][x] = 0
            for y in range(N):
                beta1[t][x] = beta1[t][x] + A1[x][y] * B1[y][O1[t + 1]] * beta1[t + 1][y]
            beta1[t][x] = c1[t][0] * beta1[t][x]
    return beta1
```

```
# Compute Gamma(x,t) and Gamma(x,y,t)
'''
The gamma function is the posterior probability of being in a particular state at a given time, given the observed
sequence. The digamma function is the posterior probability of being in a particular pair of states at two consecutive
times, given the observed sequence. The gamma and digamma functions are used in the E-step of the EM algorithm to compute
the expected sufficient statistics of the model, which are then used in the M-step to update the model parameters.
'''

def gamma_pass(alpha1, beta1, A1, B1, O1):
    gamma1 = np.zeros([T, N])
    di_gamma1 = np.zeros([T, N, N])
    for t in range(T-1):
        for x in range(N):
            gamma1[t][x] = 0
            for y in range(N):
                di_gamma1[t][x][y] = alpha1[t][x] * A1[x][y] * B1[y][O1[t + 1]] * beta1[t + 1][y]
                gamma1[t][x] = gamma1[t][x] + di_gamma1[t][x][y]
    for x in range(N):
        gamma1[T-1][x] = alpha1[T-1][x]
    return gamma1, di_gamma1
```

```
# Re-estimate A, B, pi

def re_estimate(gamma1, di_gamma1, A1, B1, pi1):
    for x in range(N):
        pi1[x] = gamma1[0][x]
    for x in range(N):
        denominator = 0
        for t in range(T-1):
            denominator = denominator + gamma1[t][x]
        for y in range(N):
            numerator = 0
            for t in range(T-1):
                numerator = numerator + di_gamma1[t][x][y]
            A1[x][y] = numerator/denominator
    for x in range(N):
        denominator = 0
        for t in range(T):
            denominator = denominator + gamma1[t][x]
        for y in range(M):
            numerator = 0
            for t in range(T):
                if O[t] == y:
                    numerator = numerator + gamma1[t][x]
            B1[x][y] = numerator/denominator
    return A1, B1, pi1


# Compute log[P(O|lambda)]

def log_prob(c1):
    logProb1 = 0
    for x in range(T):
        logProb1 = logProb1 + np.log(c1[x][0])
    logProb1 = -logProb1
    return logProb1
```

```
# Values initially

oldLogProb = -10000000
print("A: \n", A)
print("B: \n", np.concatenate((V.reshape(1, M), B), axis=0).T)
print("pi: ", pi)
print("logProb: ", oldLogProb)
```

```
A:
 [[0.47468 0.52532]
 [0.51656 0.48344]]
B:
 [['a' '0.03735' '0.03909']
 ['b' '0.03408' '0.03537']
 ['c' '0.03455' '0.03537']
 ['d' '0.03828' '0.03909']
 ['e' '0.03782' '0.03583']
 ['f' '0.03922' '0.0363']
 ['g' '0.03688' '0.04048']
 ['h' '0.03408' '0.03537']
 ['i' '0.03875' '0.03816']
 ['j' '0.04062' '0.03909']
 ['k' '0.03735' '0.0349']
 ['l' '0.03968' '0.03723']
 ['m' '0.03548' '0.03537']
 ['n' '0.03735' '0.03909']
 ['o' '0.04062' '0.03397']
 ['p' '0.03595' '0.03397']
 ['q' '0.03641' '0.03816']
 ['r' '0.03408' '0.03676']
 ['s' '0.04062' '0.04048']
 ['t' '0.03548' '0.03443']
 ['u' '0.03922' '0.03537']
 ['v' '0.04062' '0.03955']
 ['w' '0.03455' '0.03816']
 ['x' '0.03595' '0.03723']
 ['y' '0.03408' '0.03769']
 ['z' '0.03408' '0.03955']
 [' ' '0.03688' '0.03397']]
pi:  [0.525483 0.474517]
logProb:  -10000000
```

```
# After first iteration

alpha, c = alpha_pass(A, B, pi, O)
beta = beta_pass(A, B, O, c)
gamma, di_gamma = gamma_pass(alpha, beta, A, B, O)
A, B, pi = re_estimate(gamma, di_gamma, A, B, pi)
logProb = log_prob(c)

print("A: \n", A)
print("B: \n", np.concatenate((V.reshape(1, M), np.round_(B, decimals=7)), axis=0).T)
print("pi: ", np.round_(pi, decimals=7))
print("logProb: ", logProb)
```

```
A:
 [[0.48026342 0.51973658]
 [0.52229077 0.47770923]]
B:
 [['a' '0.0650532' '0.069638']
 ['b' '0.0103125' '0.0109892']
 ['c' '0.0214134' '0.0224692']
 ['d' '0.0345584' '0.0361856']
 ['e' '0.1003183' '0.0971339']
 ['f' '0.0150425' '0.0143157']
 ['g' '0.0156061' '0.0175386']
 ['h' '0.0509228' '0.0541451']
 ['i' '0.0621653' '0.0625957']
 ['j' '0.0008039' '0.000796']
 ['k' '0.0067102' '0.0064291']
 ['l' '0.0328153' '0.0315616']
 ['m' '0.0185425' '0.0189786']
 ['n' '0.0555146' '0.0594952']
 ['o' '0.0629017' '0.053836']
 ['p' '0.0195192' '0.0188993']
 ['q' '0.0005581' '0.000602']
 ['r' '0.0461575' '0.0510746']
 ['s' '0.0488464' '0.0499162']
 ['t' '0.0694392' '0.0690799']
 ['u' '0.0215736' '0.0199424']
 ['v' '0.0114356' '0.0114243']
 ['w' '0.0167326' '0.0189729']
```

```
       ['x' '0.004376' '0.0046446']
       ['y' '0.0144714' '0.0164334']
       ['z' '0.000347' '0.0004131']
       [' ' '0.1938626' '0.1824895']]
      pi:  [0.5333767 0.4666233]
      logProb:  -330336.61639803863


# Output
```