In [11]:
```
import numpy as np
with open('/Users/adilrhoulam/Desktop/mytext.txt') as f:
    hk = f.read()
F = [ord(s) for s in hk]
F = list(map(lambda x: 27 if x<33 else x,F))
F = list(map(lambda x: x - 64 if (x>64 and x<91) else x,F))
F = list(map(lambda x: x - 96 if (x>96 and x<123) else x,F))
F = list(filter(lambda x: x<27,F))
F2 = np.concatenate(([0],F[:-1]))+np.array(F)
G = []
for i in range(len(F2)):
    if F2[i]<54:
        G.append(F[i])
G1 = list(map(lambda x: x - 1 ,G))

class dynamic_programming:
    def __init__(self,G,n,k,a1,ak):
        self.G = G
        self.n = n
        self.k = k
        self.a1 = ord(a1.lower())-96-1
        self.ak = ord(ak.lower())-96-1
        self.P1 = np.zeros((26),dtype = 'int')
        self.P2 = np.zeros((26,26),dtype = 'int')
        self.P3 = np.zeros((26,26,26),dtype = 'int')
        self.P4 = np.zeros((26,26,26,26),dtype = 'int')

    def n_gram(self):
        tmp = 0
        for i in range(len(self.G)-3):
            tmp = tmp+1
            self.P1[self.G[i]] = self.P1[self.G[i]]+1
            self.P2[self.G[i],self.G[i+1]] = self.P2[self.G[i],self.G[i+1]]+1
            self.P3[self.G[i],self.G[i+1],self.G[i+2]] = \
                    self.P3[self.G[i],self.G[i+1],self.G[i+2]]+1
            self.P4[self.G[i],self.G[i+1],self.G[i+2],self.G[i+3]] = \
                    self.P4[self.G[i],self.G[i+1],self.G[i+2],self.G[i+3]]+1
        self.P1 = self.P1/tmp
        self.P2 = self.P2/tmp
        self.P3 = self.P3/tmp
        self.P4 = self.P4/tmp

    def initialization(self):
        f1 = np.zeros(tuple(26 for i in range(self.n)))
        if self.n == 2:
            prob = self.P2[self.a1]
            for b2 in range(26):
                for b3 in range(26):
                    f1[b2,b3] = -np.log(prob[b2]*self.P2[b2,b3])
            h2 = np.min(f1,axis=0)
            phi2 = np.argmin(f1,axis=0)
            return h2,phi2
        elif self.n == 3:
            prob = self.P3[self.a1]
            for b2 in range(26):
                for b3 in range(26):
                    for b4 in range(26):
                        f1[b2,b3,b4] = -np.log(prob[b2,b3]*self.P3[b2,b3,b4])
            h2 = np.min(f1,axis=0)
            phi2 = np.argmin(f1,axis=0)
            return h2,phi2
        else:
            raise('n must be 2 or 3')
    #loop for p from 2 to k-n
    def p_th_loop(self,h_p_minus_1):
        fp = np.zeros(tuple(26 for i in range(self.n)))
        summ = np.zeros(tuple(26 for i in range(self.n)))
        if self.n == 2:
            for b2 in range(26):
                for b3 in range(26):
                    fp[b2,b3] = -np.log(self.P2[b2,b3])
            for b3 in range(26):
                summ[:,b3] = h_p_minus_1 + fp[:,b3]
            h_p = np.min(summ,axis=0)
            phi_p = np.argmin(summ,axis=0)
            return h_p,phi_p
        elif self.n == 3:
            for b2 in range(26):
                for b3 in range(26):
                    for b4 in range(26):
                        fp[b2,b3,b4] = -np.log(self.P3[b2,b3,b4])
                for b4 in range(26):
                    summ[:,:,b4] = h_p_minus_1 + fp[:,:,b4]
            h_p = np.min(summ,axis=0)
            phi_p = np.argmin(summ,axis=0)
            return h_p,phi_p
        else:
            raise('n must be 2 or 3')

    def final_loop(self,h_k_minus_n):
        fnk1 = np.zeros(tuple(26 for i in range(self.n)))
        summ = np.zeros(tuple(26 for i in range(self.n)))
        if self.n == 2:
            for b2 in range(26):
                for b3 in range(26):
                    if b3 == self.ak:
                        fnk1[b2,self.ak] = -np.log(self.P2[b2,self.ak])
                    else:
                        fnk1[b2,b3] = 9999999
            for b3 in range(26):
                summ[:,b3] = h_k_minus_n + fnk1[:,b3]
            h_nk1 = np.min(summ,axis=0)
            phi_nk1 = np.argmin(summ,axis=0)
            return h_nk1,phi_nk1
        elif self.n == 3:
            for b2 in range(26):
                for b3 in range(26):
                    for b4 in range(26):
                        if b4 == self.ak:
                            fnk1[b2,b3,self.ak] = -np.log(self.P3[b2,b3,self.ak])
                        else:
                            fnk1[b2,b3,b4] = 9999999

                for b4 in range(26):
                    summ[:,:,b4] = h_k_minus_n + fnk1[:,:,b4]
            h_nk1 = np.min(summ,axis=0)
            phi_nk1 = np.argmin(summ,axis=0)
            return h_nk1,phi_nk1
        else:
            raise('n must be 2 or 3')

    def most_prob(self,h_nk1,phi_history):
        most_probable = []
        if self.n == 2:
            h = np.min(h_nk1)
            lambda_nk1 = np.argmin(h_nk1)
            most_probable.append(lambda_nk1)
            c = len(phi_history)
            for _ in range(c):
                lambda_nk1 = phi_history[c-1-_][lambda_nk1]
                most_probable.append(lambda_nk1)
            return h,most_probable
        if self.n == 3:
            h = np.min(h_nk1)
            axis1 = np.argmin(np.min(h_nk1,axis=0))
            axis0 = np.argmin(h_nk1,axis=0)[axis1]
            lambda_nk1 = tuple((axis0,axis1))
            most_probable.append(lambda_nk1)
            c = len(phi_history)
            for _ in range(c):
                tmp = axis0
                axis0 = phi_history[c-1-_][tmp,axis1]
                axis1 = tmp
                lambda_nk1 = tuple((axis0,axis1))
                most_probable.append(lambda_nk1)
            return h,most_probable
        else:
            raise('n must be 2 or 3')



if __name__ == "__main__":
    #************************************
    def return_best_word(n,k,G,a1,ak):
        phi_history = []
        dp = dynamic_programming(G,n,k,a1,ak)
        dp.n_gram()
        h2,phi2 = dp.initialization()
        phi_history.append(phi2)
        hp = h2
        for p in range(3,k-n+1):
            hp,phi_p = dp.p_th_loop(hp)
            phi_history.append(phi_p)
        h_nk1,phi_nk1 = dp.final_loop(hp)
        phi_history.append(phi_nk1)
        h,most_probable = dp.most_prob(h_nk1,phi_history)
        if n == 2:
            most_probable.reverse()
            word = a1+''.join(list(map(lambda x: chr(x+97),most_probable)))
            print('The most probable word for a1 =',a1,', ak =',ak,', k =',k,' and n =',n,' is:
 ', word)
        if n == 3:
            most_probable.reverse()
            word = a1+''.join([chr(most_probable[0][0])]+list(map(lambda x: chr(x[1]+97),most_p
robable)))
            print('The most probable word for a1 =',a1,', ak =',ak,', k =',k,' and n =',n,' is:
 ', word)

    return_best_word(2,11,G1,'e','y')
    return_best_word(2,10,G1,'j','m')
    return_best_word(2,14,G1,'c','n')
    return_best_word(3,11,G1,'e','y')
    return_best_word(3,10,G1,'j','m')
    return_best_word(3,14,G1,'c','n')
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:47: RuntimeWarning: divide by z
ero encountered in log
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:69: RuntimeWarning: divide by z
ero encountered in log
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:95: RuntimeWarning: divide by z
ero encountered in log
```

```
The most probable word for a1 = e , ak = y , k = 11  and n = 2  is:  ethethethey
The most probable word for a1 = j , ak = m , k = 10  and n = 2  is:  jonthethem
The most probable word for a1 = c , ak = n , k = 14  and n = 2  is:  cathenthenthen
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:56: RuntimeWarning: divide by z
ero encountered in log
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:79: RuntimeWarning: divide by z
ero encountered in log
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:108: RuntimeWarning: divide by
zero encountered in log
```

```
The most probable word for a1 = e , ak = y , k = 11  and n = 3  is:  e□herethery
The most probable word for a1 = j , ak = m , k = 10  and n = 3  is:  j□sthethem
The most probable word for a1 = c , ak = n , k = 14  and n = 3  is:  c□ntheretheren
```

We see that the 3 words predicted given n equal to 2 or 3 are not meaningfull, we can see that the predicted substring when