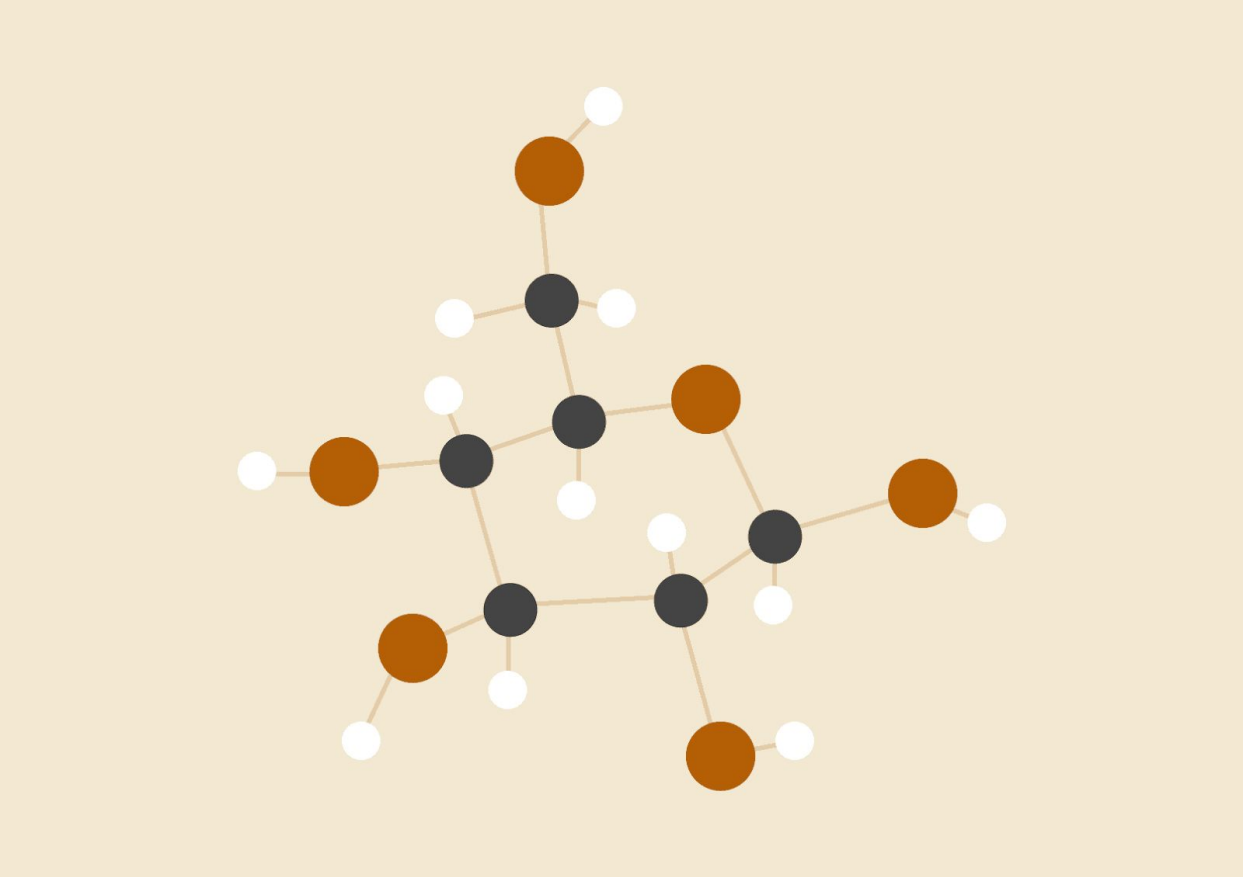
ΕΡΓΑΣΙΑ 1

https://github.com/IsabellaPap/TOIproject



Ερρίκος Φοίβος Καββαδίας: 2019070

Παπαγεωργίου Ισαβέλλα: 2019235

15.11.2020-29.11.2020

Information Theory|Ionian University

Στην αρχή κάθε λύσης υπάρχουν τα εκτελέσιμα που αντιστοιχούν στο ερώτημα, καθώς και τα αρχεία που εξάγει, ακόμα και αν αυτά δεν ζητούνται . Ο κώδικας μας μπορεί να βρεθεί εύκολα και στο github στο link που βρίσκεται στο εξώφυλλο.

Ερώτημα 1

Υπολογίστε την πιθανότητα εμφάνισης των γραμμάτων στην ελληνική γλώσσα χρησιμοποιώντας το κείμενο “NEO_SYNTAGMA.txt”, αφού αφαιρέσετε από αυτό όσους χαρακτήρες δεν ανήκουν στα γράμματα του ελληνικού αλφάβητου

Λύση:

Κώδικας: erwtima_1.py Output: NEO_SYNTAGMA_AB.txt, NEO_SYNTAGMA_KEF.txt

Πιθανότητα εμφάνισης γραμμάτων στην ελληνική (pdf):

```
{'Α': 0.10435005555080762, 'Β': 0.0067515596957525, 'Γ':
0.016618626224642805, 'Δ': 0.020456682024069427, 'Ε':
0.07780997739120976, 'Ζ': 0.003993442673897336, 'Η':
0.05896154951791222, 'Θ': 0.011180085618167834, 'Ι':
0.09310781518285151, 'Κ': 0.03970911577099082, 'Λ':
0.023440110013907126, 'Μ': 0.03018390036593609, 'Ν':
0.05604027627786281, 'Ξ': 0.0041954456107092634, 'Ο':
0.10884850556673478, 'Π': 0.04182237726379253, 'Ρ':
0.04810000699240935, 'Σ': 0.0792550753237874, 'Τ':
0.07989993085284086, 'Υ': 0.051860369354600616, 'Φ':
0.008631740876848133, 'Χ': 0.0080801174724771, 'Ψ':
0.0017170249629013838, 'Ω': 0.024986209414890725}
```

Κώδικας(προεπισκόπηση - μη-εκτελέσιμος):

```
with open('NEO_SYNTAGMA.txt','r') as f1:
   data = f1.read()
```

```
with open('NEO_SYNTAGMA_AB.txt','w') as f2:

   letters = {'α':0,'Α':0,'Ἀ':0,'ά':0,'β':0,'Β':0,...
...'Υ':0,'Ύ':0,'ύ':0,'ϋ':0,'φ':0,'Φ':0,'χ':0,'Χ':0,'ψ':0,'Ψ':0,'ω':0,'Ω':0
,'Ὼ':0,'ώ':0}

   for i in data:
       if i in letters.keys():
           f2.write(i)

def alphabetGR_count_distribution(text_file):

   with open (text_file, 'r') as f:
       data = f.read()

   letters =
{'Α':0,'Β':0,'Γ':0,'Δ':0,'Ε':0,'Ζ':0,'Η':0,'Θ':0,'Ι':0,'Κ':0,'Λ':0,'Μ':
0,'Ν':0,'Ξ':0,'Ο':0,'Π':0,'Ρ':0,'Σ':0,'Τ':0,'Υ':0,'Φ':0,'Χ':0,'Ψ':0,'Ω'
:0}
   with open ('NEO_SYNTAGMA_KEF.txt','w') as f2:
       total = 0
       for i in data:
           if i in ['α','Α','ά','Ἀ']:
               total += 1
               letters['Α'] += 1
               f2.write('Α')
           elif i in ['β','Β']:
               total += 1
               letters['Β'] += 1
               f2.write('Β')
                   .
                   .
                   .

           elif i in ['ω','Ω','ώ','Ὼ']:
               total += 1
               letters['Ω'] += 1
               f2.write('Ω')

       p {'Α':0,'Β':0,...,'Ω':0}
       for l in letters.keys():
           num_letters += 1
           p[l] = letters[l]/total
       return letters, p
```

Ερώτημα 2

Υπολογίστε την εντροπία του ελληνικού αλφάβητου (ανά ένα γράμμα) αν τα γράμματα εμφανίζονται ισοπίθανα (ομοιόμορφη κατανομή u).

Λύση:

Κώδικας: erwtima_2.py

Εντροπία του ελληνικού αλφάβητου για ομοιόμορφη κατανομή u (h_equal): `4.584962500721157`

Κώδικας(προεπισκόπηση - μη-εκτελέσιμος):

```
#letters in greek alphabet
num_letters = 24
x = 1/num_letters

# u = equal distribution
u = {'Α':x, 'Β':x, 'Γ':x, 'Δ':x, 'Ε':x, 'Ζ':x, 'Η':x, 'Θ':x, 'Ι':x,
    'Κ':x, 'Λ':x, 'Μ':x, 'Ν':x, 'Ξ':x, 'Ο':x, 'Π':x, 'Ρ':x, 'Σ':x,
    'Τ':x, 'Υ':x, 'Φ':x, 'Χ':x, 'Ψ':x, 'Ω':x}

h_equal = math.log(num_letters,2)
```

Ερώτημα 3

Υπολογίστε την εντροπία του ελληνικού αλφάβητου ακολουθώντας την κατανομή p που βρήκατε στο κείμενο “NEO_SYNTAGMA_AB.txt”.

Λύση:

Κώδικας: erwtima_3.py

Εντροπία του ελληνικού αλφάβητου για κατανομή p (H): `4.0999115567520485`

Κώδικας(προεπισκόπηση - μη-εκτελέσιμος):

```
H = 0
for i in pdf.keys():
   if pdf[i] > 0:
       H += -pdf[i]*math.log(pdf[i],2)
```

Ερώτημα 4

Υπολογίστε την απόσταση “Kullback Leibler” της κατανομής p που βρήκατε από την ομοιόμορφη κατανομή u.

Λύση:

Κώδικας: erwtima_4.py

Απόσταση “Kullback Leibler” της κατανομής p (distance): 0.485050943969108

Κώδικας(προεπισκόπηση - μη-εκτελέσιμος):

```
for i in pdf.keys():
   distance += pdf[i]*math.log(pdf[i]/u[i],2)
```

Ερώτημα 5

Κωδικοποιήστε το κείμενο “NEO_SYNTAGMA_AB.txt” με χρήση κώδικα Shannon-Fano-Elias. Αποκωδικοποιήστε το κείμενο και ελέγξτε την αρτιότητα του αποκωδικοποιημένου σε σχέση με το αρχικό.

Λύση:

Κώδικας: S_F_2019070_2019235, Output: S_F_2019070_2019235.txt, S_F_2019070_2019235_DECODED.txt

Διαφορά κωδικοποιημένου μηνύματος από αρχικό(hamming_distance): 120107

Πέρα από την απόσταση hamming, εξετάζοντας το κείμενο, είναι δύσκολο κάποιος

να βγάλει νόημα καθώς τα γράμματα είναι σε τέτοια σειρά που δεν βγάζει νόημα. Κανονικά αυτό δεν θα έπρεπε να συμβαίνει, αφού ο κώδικας είναι απροθεματικός, και όπως βλέπουμε από το ερώτημα 7 παρακάτω, ο κώδικας έχει σχετικά υψηλή απόδοση.

'Αλφάβητο' : `{'Α': '00001', 'Β': '000110111', 'Γ': '0001111', 'Δ': '0010001', 'Ε': '00101', 'Ζ': '001110100', 'Η': '010000', 'Θ': '01001011', 'Ι': '01011', 'Κ': '011010', 'Λ': '0111000', 'Μ': '0111100', 'Ν': '100000', 'Ξ': '100010110', 'Ο': '10011', 'Π': '101011', 'Ρ': '101110', 'Σ': '11001', 'Τ': '11011', 'Υ': '111011', 'Φ': '11110101', 'Χ': '11111000', 'Ψ': '11111001011', 'Ω': '1111110'}`

Κώδικας(προεπισκόπηση - μη-εκτελέσιμος):

```
# CODING PART

def code_words(a):
   # a is float
   a = a - int(a)
   # now we have only the decimal part
   r = -1
   b = []
   # limit for code word 5 bits
   while a > 10**(-5):
       if a >= 2**r:
           b.append('1')
           a = a - 2**r
       else:
           b.append('0')
       r -= 1
   return ''.join(b)


def length_words(p):
   lp = []
   for i in p:
       # L(x) = ceiling of log(1/p[i],2)+1
       m_l = math.log(1/i,2)
       if m_l == int(m_l):
           lp.append(int(m_l)+1)
```

```
        else:
            lp.append(int(m_l)+2)
    return lp
# f keeps track of Sums so it can add the previous to the next
f = []
# fi contains all the results of F for every probability
fi = []
# s is the sum as we move (stored in f)
s = 0
# implementation of F function calculation
for i in range(len(p)):
    s += p[i]
    f.append(s)
    if i == 0:
        fi.append(p[i]/2)
    else:
        fi.append(f[i-1] + p[i]/2)

l = []
# i will take one by one the results of F function
for i in fi:
    # code_words converts fi results to bit sequences
    l.append(code_words(i))

# finding the length of each word
l_w = length_words(p)
c = []
for i in range(len(p)):
    # because code_words returns joined array [:l_w[i]] indicates how many bits to
use
    c.append(l[i][:l_w[i]])

def coding(mes,code):
    coded_mes = []
    for i in mes:
        coded_mes.append(code[i])
    return ''.join(coded_mes)

# Create the message

coded_mes = coding(mes_in,code)

# DECODING PART
```

```
# Decoding the message
def decoding01(mes,code):
   decoded = []
   for i in range(0,len(mes)):
       letter = ''
       # checks for every value if it matches with codeword
       # because it is a "prefix code" we will not have matches regardless of the
length we check
       for key, value in code.items():
           # for every codeword it checks if the same length matches the code
           if value == mes[i:i+len(code[key])]:
               letter = key
       i += len(letter)
         decoded.append(letter)
   return ''.join(decoded)

r_message = decoding01(coded_mes,code)

def hamming_distance(s1, s2):
   return sum(c1 != c2 for c1, c2 in zip(s1, s2))
```

Ερώτημα 6

Κωδικοποιήστε το κείμενο “NEO_SYNTAGMA_AB.txt” με χρήση κώδικα Huffman. Αποκωδικοποιήστε το κείμενο και ελέγξτε την αρτιότητα του αποκωδικοποιημένου σε σχέση με το αρχικό.

Λύση:

Κώδικας: Huff_2019070_2019235.py, Output: Huff_2019070_2019235.txt, Huff_2019070_2019235_DECODED.txt

Διαφορά κωδικοποιημένου μηνύματος από αρχικό(hamming_distance): 119843

Ξανά, για αυτό το ερώτημα ισχύουν οι παρατηρήσεις του προηγούμενου ερωτήματος. Και εδώ έχουμε ένα αποκωδικοποιημένο κείμενο χωρίς συνοχή.

‘Αλφάβητο’ : `{'Α': '010', 'Ο': '011', ' ': '000', 'Σ': '1100', 'Τ': '1101', 'Ε': '1011', 'Ν': '1000', 'Η': '1001', 'Υ':`

'0011', 'Ρ': '11111', 'Κ': '11100', 'Π': '11101', 'Μ': '10100', 'Ω': '00100', 'Δ': '111100', 'Λ': '111101', 'Γ': '101010', 'Θ': '001010', 'Φ': '1010110', 'Β': '0010110', 'Χ': '0010111', 'Ξ': '10101110', 'Ψ': '101011110', 'Ζ': '101011111'}

Κώδικας(προεπισκόπηση - μη-εκτελέσιμος):

```
def huffman(p):

   # Base case of only two symbols, assign 0 or 1 arbitrarily
   if(len(p) == 2):
       return dict(zip(p.keys(), ['0', '1']))


   p_prime = p.copy()
   a1, a2 = lowest_prob_pair(p)
   p1, p2 = p_prime.pop(a1), p_prime.pop(a2)
   p_prime[a1 + a2] = p1 + p2


   c = huffman(p_prime)
   ca1a2 = c.pop(a1 + a2)
   c[a1], c[a2] = ca1a2 + '0', ca1a2 + '1'

   return c

def lowest_prob_pair(p):

   assert(len(p) >= 2)

   sorted_p = sorted(p.items(), key=lambda x: x[1])
   return sorted_p[0][0], sorted_p[1][0]


def coding(mes,code):
   coded_mes = []
   for i in mes:
       coded_mes.append(code[i])
   return ''.join(coded_mes)

# Create the message
coded_mes = coding(mes_in,huffman(pdf))
```

```
# DECODING PART

# Decoding the message
def decoding01(mes,code):
   decoded = []
   for i in range(0,len(mes)):
       letter = ''
       # checks for every value if it matches with codeword
       # because it is a "prefix code" we will not have matches regardless of
the length we check
       for key, value in code.items():
           # for every codeword it checks if the same length matches the code
           if value == mes[i:i+len(code[key])]:
               letter = key
       i += len(letter)
        decoded.append(letter)
   return ''.join(decoded)

r_message = decoding01(coded_mes,huffman(pdf))


def hamming_distance(s1, s2):
   return sum(c1 != c2 for c1, c2 in zip(s1, s2))
```

Ερώτημα 7

Υπολογίστε την απόδοση του κάθε κώδικα που δημιουργήσατε,καθώς και την απόδοση κώδικα που θα κωδικοποιεί κάθε γράμμα με κωδικολέξη μήκους 8-bit.

Λύση:

Κώδικας: erwtima_7.py, Huff_2019070_2019235.py, S_F_2019070_2019235

Απόδοση κώδικα κωδικοποίησης (Eff = h/l_sfe):
Shannon-Fano: `0.7216115691982102`
Huffman: `0.7931863657396302` (περιμέναμε αρκετά μεγαλύτερη απόδοση με βάση τη θεωρία, όπου έχουμε δει παράδειγμα με απόδοση >90%)

Κώδικας 8-bit: `0.5124889445940061`

Κώδικας(προεπισκόπηση - μη-εκτελέσιμος):

```
# EFFICIENCY
h = 0
l_sfe = 0
for i in range(len(p)):
   h -= p[i] * math.log(p[i],2)
   l_sfe += p[i] * len(c[i])
print('H(X) = ',h,'L(X) = ',l_sfe,'Eff =',h/l_sfe)

# CODING PART

def code_words(a):
   # a is float
   a = a - int(a)
   # now we have only the decimal part
   r = -1
   b = []
   # limit for code word 5 bits
   while a > 10**(-5):
       if a >= 2**r:
           b.append('1')
           a = a - 2**r
       else:
           b.append('0')
       r -= 1
   return ''.join(b)

# f keeps track of Sums so it can add the previous to the next
f = []
# fi contains all the results of F for every probability
fi = []
# s is the sum as we move (stored in f)
s = 0
# implementation of F function calculation
for i in range(len(p)):
   s += p[i]
   f.append(s)
   if i == 0:
       fi.append(p[i]/2)
   else:
       fi.append(f[i-1] + p[i]/2)
```

```
l = []
# i will take one by one the results of F function
for i in fi:
   # code_words converts fi results to bit sequences
   l.append(code_words(i))

# finding the length of each word = 8 bits
l_w = 8
c = []
for i in range(len(p)):
   # because code_words returns joined array [:l_w[i]] indicates how many bits to
use
   c.append(l[i][:8])
```

Ερώτημα 8

Αναφέρατε το μέγεθος του κειμένου “NEO_SYNTAGMA_AB.txt”όπως είναι αποθηκευμένο στον υπολογιστή σας, καθώς και το μέγεθος που έχει το κωδικοποιημένο κείμενο με τους κώδικες των(5) και (6) υπολογισμένο με ένα bit για κάθε χαρακτήρα (0 ή 1) και δώστε το ποσοστό συμπίεσης που επιτεύχθηκε σε κάθε περίπτωση.

Λύση:

Κώδικας: erwtima_8.py

Μέγεθος κωδικοποιημένου μηνύματος σε bits (size_of_X):

```
Size of Shannon Fano coded message in bits =  731285
```

```
Size of Huffman coded message in bits =  532556
```

Ποσοστό συμπίεσης (X_ratio)

```
Shannon-Fano: 35.502305051305356 %
```

```
Huffman: 25.854441932903004 %
```

Κώδικας(προεπισκόπηση - μη-εκτελέσιμος):

```
import os
import math

size_of_og = os.path.getsize('NEO_SYNTAGMA_AB.txt') *8

size_of_SF = os.path.getsize('S_F_2019070_2019235.txt')

size_of_Huff = os.path.getsize('Huff_2019070_2019235.txt')

SF_ratio = (size_of_SF/size_of_og) *100
Huff_ratio = (size_of_Huff/size_of_og) *100
```