ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λύσεις 1ης Σειράς Ασκήσεων για το μάθημα "Υπολογιστική Κρυπτογραφία"

Δημήτριος Βασιλείου
03119830
9ο Εξάμηνο

ΑΣΚΗΣΗ 1

Λύσαμε την άσκηση αυτή με 2 τρόπους. Ο ένας αφορά στην κατασκευή προγράμματος καθαρά βοηθητικού, που δεν λειτουργεί για κάθε κείμενο, καθώς χρειάστηκαν και υποθέσεις για τα γράμματα με βάση το ίδιο το κείμενο. Ο δεύτερος τρόπος είναι πιο αυτοματοποιημένος, λειτουργεί για κάθε κείμενο, ωστόσο ο κώδικάς του είναι από το διαδίκτυο.

<u>1ος τρόπος:</u>

Ο κώδικας που αναπτύξαμε σε γλώσσα Python είναι ο ακόλουθος:

```python
from collections import defaultdict
import re

english_freqs = [ {'A': 8.5}, {'B': 2.07}, {'C': 4.54}, {'D': 3.38}, {'E': 11.16},
{'F': 1.81}, {'G': 2.47}, {'H': 3},
                  {'I': 7.54}, {'J': 0.2}, {'K': 1.1}, {'L': 5.49}, {'M': 3.01}, {'N':
6.65}, {'O': 7.16}, {'P': 3.17},
                  {'Q': 0.2}, {'R': 7.58}, {'S': 5.74}, {'T': 6.95}, {'U': 3.63}, {'V':
1.01}, {'W': 1.29}, {'X': 0.29},
                  {'Y': 1.78}, {'Z': 0.27} ]

english_freqs = [{'E': 11.16}, {'A': 8.5}, {'R': 7.58}, {'I': 7.54}, {'O': 7.16}, {'T':
6.95}, {'N': 6.65}, {'S': 5.74},
                 {'L': 5.49}, {'C': 4.54}, {'U': 3.63}, {'D': 3.38}, {'P': 3.17}, {'M':
3.01}, {'H': 3}, {'G': 2.47}, {'B': 2.07},
                 {'F': 1.81}, {'Y': 1.78}, {'W': 1.29}, {'K': 1.1}, {'V': 1.01}, {'X':
0.29}, {'Z': 0.27}, {'J': 0.2}, {'Q': 0.2}]

input = []
input_length = 0
input_freqs = [ {'A': 0}, {'B': 0}, {'C': 0}, {'D': 0}, {'E': 0}, {'F': 0}, {'G': 0},
{'H': 0},
                {'I': 0}, {'J': 0}, {'K': 0}, {'L': 0}, {'M': 0}, {'N': 0}, {'O': 0},
{'P': 0},
                {'Q': 0}, {'R': 0}, {'S': 0}, {'T': 0}, {'U': 0}, {'V': 0}, {'W': 0},
{'X': 0},
                {'Y': 0}, {'Z': 0} ]
letter_counter = 0


doubles = []
triples = []

def decrypt(input_file):
    global letter_counter
    global input_freqs
    with open(input_file, 'r') as file:
        input = list(file.read())

    for i in range (0, len(input)):
        # check if current input character is letter
        if input[i].isalpha():
            current_letter = input[i]
            # iterate the frequencies to find the element with the current_letter
            for letter_freq in input_freqs:
                if current_letter in letter_freq:
                    # increase the frequency by 1
                    letter_freq[current_letter] += 1
            letter_counter += 1

    # sort input_freqs on descending order
    input_freqs = sorted(input_freqs, key = lambda x:list(x.values())[0], reverse=True)
    print(input_freqs)
```

```python
def collect_bigrams(input_file):
    with open(input_file, 'r') as file:
        lines = file.readlines()
    input_text =''.join(lines)

    words = re.findall(r'\b[A-Za-z]+\b', input_text)
    bigram_freqs = defaultdict(int)

    for word in words:
        for i in range (0, len(word)):
            if (len(word) > 1 and (i + 1 < len(word))):
                bigram = word[i] + word[i + 1]
                bigram_freqs[bigram] += 1
    bigram_freqs = sorted(bigram_freqs.items(), key=lambda x: x[1], reverse=True)
    return bigram_freqs


def collect_trigrams(input_file):
    with open(input_file, 'r') as file:
        lines = file.readlines()
    input_text =''.join(lines)

    words = re.findall(r'\b[A-Za-z]+\b', input_text)
    trigram_freqs = defaultdict(int)

    for word in words:
        for i in range (0, len(word)):
            if (len(word) > 2 and (i + 2 < len(word))):
                trigram = word[i] + word[i + 1] + word[i + 2]
                trigram_freqs[trigram] += 1
    trigram_freqs = sorted(trigram_freqs.items(), key=lambda x: x[1], reverse=True)
    return trigram_freqs

def find_words_with_p(input_file):
    with open(input_file, 'r') as file:
        lines = file.readlines()
    input_text =''.join(lines)

    words = re.findall(r'\b[A-Za-z]+\b', input_text)
    res = []
    allowed_letters = set('UKVRQWLCGI')
    for word in words:
        for letter in word:
            #if 'U' in word or 'K' in word or 'V' in word or 'R' in word or 'Q' in word
or 'W' in word or 'L' in word or 'C' in word or 'G' in word and word not in res and
len(word) <= 3:
            if letter in allowed_letters and word not in res and len(word) < 6:
                res.append(word)
                break
    return res

def replace_letters(input_file, letter_mapping):
    with open(input_file, 'r') as file:
        lines = file.readlines()
    input_text =''.join(lines)
    # Convert the list of dictionaries into a dictionary
    mapping_dict = {k: v for mapping in letter_mapping for k, v in mapping.items()}

    # Replace letters in the text using the mapping
    replaced_text = ''.join(mapping_dict.get(char, char) for char in  input_text)

    return replaced_text


input_file = "cipher.txt"
```

```
decrypt(input_file)
print("\n\n")
print(collect_bigrams(input_file))
print("\n\n")
print(collect_trigrams(input_file))
print("\n\n")
print(find_words_with_p(input_file))
print("\n\n")
# MEROS -> ZEROS αρα M->Z. Το S δεν παει καπου γτ δεν εμφανιζεται πουθενα
letter_mapping = [{'U': 'E'}, {'K': 'T'}, {'V': 'H'}, {'R': 'R'}, {'Q': 'O'}, {'W':
'A'}, {'L': 'L'}, {'C': 'I'}, {'G': 'S'}, {'I': 'P'}, {'D': 'N'}, {'N': 'U'}, {'B':
'M'}, {'A': 'B'},
                  {'X': 'V'}, {'Z': 'W'}, {'H': 'C'}, {'T': 'X'}, {'J': 'G'}, {'O':
'Y'}, {'Y': 'D'}, {'P': 'F'}, {'E': 'J'}, {'F': 'Q'}, {'M': 'Z'}]
print(replace_letters(input_file, letter_mapping))
```

- Η συνάρτηση collect_bigrams συλλέγει σε μια λίστα όλα τα διγράμματα του κειμένου και η συνάρτηση collect_trigrams συλλέγει όλα τα τριγράμματα του κειμένου. Με βάση τις συχνότητες εμφάνισης διγραμμάτων και τριγραμμάτων καταλήγουμε στην αντιστοιχία ότι $KVU \rightarrow THE$.
- Παρατηρώντας το κείμενο βλέπουμε επίσης ότι $R \rightarrow R$ και $Q \rightarrow O$ αφού υπάρχει στο κείμενο το KQ και γνωρίζοντας ότι $K \rightarrow T$, η μοναδική λέξη που αρχίζει από T και έχει δύο γράμματα είναι το TO.
- Παρατηρούμε ακόμα, πως υπάρχει η λέξη με τρία γράμματα WLL. Επίσης με μεγάλη πιθανότητα το W είναι είτε I είτε A, γιατί εμφανίζεται μόνο του σαν λέξη. Υποθέτοντας ότι $L \rightarrow L$ προκύπτει ότι $W \rightarrow A$ και $WLL \rightarrow ALL$.
- Επίσης, θα είναι $C \rightarrow I$ καθώς το C είναι το μόνο μαζί με W που εμφανίζεται ως λέξη μόνο του.
- Επιπλέον, αντικαθιστούμε $G \rightarrow S$, καθώς στο κείμενο υπάρχει $HLWGG$ άρα ταιριάζει να είναι $GG \rightarrow SS$. Τέλος, κάνουμε την αντικατάσταση $I \rightarrow H$, διότι το $C \rightarrow I$ και υπάρχει η λέξη IC στο κείμενο, που είναι λογικό να είναι $IC \rightarrow HI$.
- Σε αυτό το σημείο έχουμε βρει τις αντικαταστάσεις των γραμμάτων $U, K, V, R, Q, W, L, C, G, I$.
- Η συνάρτηση find_words_with_p επιστρέφει μια λίστα με τις λέξεις του κειμένου που περιέχουν τουλάχιστον ένα χαρακτήρα εκ των παραπάνω που έχουμε βρει, ώστε να μας βοηθήσει να βρούμε και τις αντιστοιχίσεις των υπόλοιπων χαρακτήρων.
- Η συνάρτηση replace_letters παίρνει ως είσοδο ένα dictionary με τις αντιστοιχίες των γραμμάτων και επιστρέφει την αποκρυπτογράφηση του κειμένου με βάση αυτές τις αντιστοιχίες. Αν η αντιστοίχιση δεν είναι πλήρης, δηλαδή αν δεν έχουμε αντιστοιχίσει ακόμα όλα τα γράμματα, αποκρυπτογραφούνται μόνο τα γράμματα που ξέρουμε την αντιστοιχία τους. Έτσι, δίνοντας ως είσοδο τις αντιστοιχίες για τα γράμματα $U, K, V, R, Q, W, L, C, G, I$ που έχουμε βρει, και στη συνέχεια δοκιμάζοντας, καταλήγουμε σε αντιστοιχία όλων των γραμμάτων και συνεπώς στο αποκρυπτογραφημένο κείμενο.

  Το αποκρυπτογραφημένο κείμενο είναι το ακόλουθο:

```
THE COMPUTABLE NUMBERS MAY BE DESCRIBED BRIEFLY AS THE REAL NUMBERS WHOSE
EXPRESSIONS AS A DECIMAL ARE CALCULABLE BY FINITE MEANS. ALTHOUGH THE SUBJECT
OF THIS PAPER IS OSTENSIBLY THE COMPUTABLE NUMBERS. IT IS ALMOST EQUALLY
EASY TO DEFINE AND INVESTIGATE COMPUTABLE FUNCTIONS OF AN INTEGRAL VARIABLE
OR A REAL OR COMPUTABLE VARIABLE, COMPUTABLE PREDICATES, AND SO FORTH. THE
FUNDAMENTAL PROBLEMS INVOLVED ARE, HOWEVER, THE SAME IN EACH CASE, AND I HAVE
CHOSEN THE COMPUTABLE NUMBERS FOR EXPLICIT TREATMENT AS INVOLVING THE LEAST
CUMBROUS TECHNIQUE. I HOPE SHORTLY TO GIVE AN ACCOUNT OF THE RELATIONS OF THE
COMPUTABLE NUMBERS, FUNCTIONS, AND SO FORTH TO ONE ANOTHER. THIS WILL INCLUDE A
DEVELOPMENT OF THE THEORY OF FUNCTIONS OF A REAL VARIABLE EXPRESSED IN TERMS
OF COMPUTABLE NUMBERS. ACCORDING TO MY DEFINITION, A NUMBER IS COMPUTABLE
IF ITS DECIMAL CAN BE WRITTEN DOWN BY A MACHINE. I GIVE SOME ARGUMENTS WITH
THE INTENTION OF SHOWING THAT THE COMPUTABLE NUMBERS INCLUDE ALL NUMBERS
WHICH COULD NATURALLY BE REGARDED AS COMPUTABLE. IN PARTICULAR, I SHOW THAT
CERTAIN LARGE CLASSES OF NUMBERS ARE COMPUTABLE. THEY INCLUDE, FOR INSTANCE,
THE REAL PARTS OF ALL ALGEBRAIC NUMBERS, THE REAL PARTS OF THE ZEROS OF THE
BESSEL FUNCTIONS, THE NUMBERS PI, E, ETC. THE COMPUTABLE NUMBERS DO NOT,
HOWEVER, INCLUDE ALL DEFINABLE NUMBERS, AND AN EXAMPLE IS GIVEN OF A DEFINABLE
NUMBER WHICH IS NOT COMPUTABLE. ALTHOUGH THE CLASS OF COMPUTABLE NUMBERS
IS SO GREAT, AND IN MANY WAYS SIMILAR TO THE CLASS OF REAL NUMBERS, IT IS
NEVERTHELESS ENUMERABLE. I EXAMINE CERTAIN ARGUMENTS WHICH WOULD SEEM TO PROVE
THE CONTRARY. BY THE CORRECT APPLICATION OF ONE OF THESE ARGUMENTS, CONCLUSIONS
ARE REACHED WHICH ARE SUPERFICIALLY SIMILAR TO THOSE OF GODEL. THESE RESULTS
HAVE VALUABLE APPLICATIONS. IN PARTICULAR, IT IS SHOWN THAT THE HILBERTIAN
ENTSCHEIDUNGSPROBLEM CAN HAVE NO SOLUTION.
```

Η αντιστοίχιση των γραμμάτων (κλειδί) είναι η εξής:

```
letter_mapping = [{'U': 'E'}, {'K': 'T'}, {'V': 'H'}, {'R': 'R'}, {'Q': 'O'}, {'W': 'A'}, {'L': 'L'}, {'C': 'I'},
                  {'G': 'S'}, {'I': 'P'}, {'D': 'N'}, {'N': 'U'}, {'B': 'M'}, {'A': 'B'},
                  {'X': 'V'}, {'Z': 'W'}, {'H': 'C'}, {'T': 'X'}, {'J': 'G'}, {'O': 'Y'},
                  {'Y': 'D'}, {'P': 'F'}, {'E': 'J'}, {'F': 'Q'}, {'M': 'Z'}]
```

2ος τρόπος:

Η βασική ιδέα αυτής της υλοποίησης έγκειται στην brute force επίθεση πάνω στο κλειδί. Ξεκινάμε με ένα τυχαίο κλειδί και κάνουμε μεταθέσεις. Βρίσκουμε τα plaintexts με βάση αυτές τις μεταθέσεις και υπολογίζουμε το σκορ κάθε πιθανού plaintext με βάση μια fitness function. Όσο καλυτερεύει το σκορ, τόσο προσεγγίζουμε το κλειδί.

Έχουμε δύο αρχεία, το ask1_auto.py και το ngram_score.py.

ask1_auto.py :
Παραθέτουμε τον κώδικα:

```
from pycipher import SimpleSubstitution as SimpleSub
import random
import re
from ngram_score import ngram_score

fitness = ngram_score('english_quadgrams.txt') # load our quadgram statistics

ctext='''KVU HQBINKWALU DNBAURG BWO AU YUGHRCAUY ARCUPLO WG KVU RUWL DNBAURG ZVQGU
UTIRUGGCQDG WG W YUHCBWL WRU HWLHNLWALU AO PCDCKU BUWDG. WLKVQNJV KVU GNAEUHK
QP KVCG IWIUR CG QGKUDGCALO KVU HQBINKWALU DNBAURG. CK CG WLBQGK UFNWLLO
UWGO KQ YUPCDU WDY CDXUGKCJWKU HQBINKWALU PNDHKCQDG QP WD CDKUJRWL XWRCWALU
QR W RUWL QR HQBINKWALU XWRCWALU, HQBINKWALU IRUYCHWKUG, WDY GQ PQRKV. KVU
PNDYWBUDKWL IRQALUBG CDXQLXUY WRU, VQZUXUR, KVU GWBU CD UWHV HWGU, WDY C VWXU
```

```
HVQGUD KVU HQBINKWALU DNBAURG PQR UTILCHCK KRUWKBUDK WG CDXQLXCDJ KVU LUWGK
HNBARQNG KUHVDCFNU. C VQIU GVQRKLO KQ JCXU WD WHHQNDK QP KVU RULWKCQDG QP KVU
HQBINKWALU DNBAURG, PNDHKCQDG, WDY GQ PQRKV KQ QDU WDQKVUR. KVCG ZCLL CDHLNYU W
YUXULQIBUDK QP KVU KVUQRO QP PNDHKCQDG QP W RUWL XWRCWALU UTIRUGGUY CD KURBG
QP HQBINKWALU DNBAURG. WHHQRYCDJ KQ BO YUPCDCKCQD, W DNBAUR CG HQBINKWALU
CP CKG YUHCBWL HWD AU ZRCKKUD YQZD AO W BWHVCDU. C JCXU GQBU WRJNBUDKG ZCKV
KVU CDKUDKCQD QP GVQZCDJ KVWK KVU HQBINKWALU DNBAURG CDHLNYU WLL DNBAURG
ZVCHV HQNLY DWKNRWLLO AU RUJWRYUY WG HQBINKWALU. CD IWRKCHNLWR, C GVQZ KVWK
HURKWCD LWRJU HLWGGUG QP DNBAURG WRU HQBINKWALU. KVUO CDHLNYU, PQR CDGKWDHU,
KVU RUWL IWRKG QP WLL WLJUARWCH DNBAURG, KVU RUWL IWRKG QP KVU MURQG QP KVU
AUGGUL PNDHKCQDG, KVU DNBAURG IC, U, UKH. KVU HQBINKWALU DNBAURG YQ DQK,
VQZUXUR, CDHLNYU WLL YUPCDWALU DNBAURG, WDY WD UTWBILU CG JCXUD QP W YUPCDWALU
DNBAUR ZVCHV CG DQK HQBINKWALU. WLKVQNJV KVU HLWGG QP HQBINKWALU DNBAURG
CG GQ JRUWK, WDY CD BWDO ZWOG GCBCLWR KQ KVU HLWGG QP RUWL DNBAURG, CK CG
DUXURKVULUGG UDNBURWALU. C UTWBCDU HURKWCD WRJNBUDKG ZVCHV ZQNLY GUUB KQ IRQXU
KVU HQDKRWRO. AO KVU HQRRUHK WIILCHWKCQD QP QDU QP KVUGU WRJNBUDKG, HQDHLNGCQDG
WRU RUWHVUY ZVCHV WRU GNIURPCHCWLLO GCBCLWR KQ KVQGU QP JQYUL. KVUGU RUGNLKG
VWXU XWLNWALU WIILCHWKCQDG. CD IWRKCHNLWR, CK CG GVQZD KVWK KVU VCLAURKCWD
UDKGHVUCYNDJGIRQALUB HWD VWXU DQ GQLNKCQD.
'''
ctext = re.sub('[^A-Z]','',ctext.upper())

maxkey = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
maxscore = -99e9
parentscore,parentkey = maxscore,maxkey[:]
print ("Substitution Cipher solver, you may have to wait several iterations")
print ("for the correct result.")
# keep going until we are killed by the user
i = 0
while 1:
    i = i+1
    random.shuffle(parentkey)
    deciphered = SimpleSub(parentkey).decipher(ctext)
    parentscore = fitness.score(deciphered)
    if (parentscore > maxscore):
        count = 0
        while count < 1000:
            a = random.randint(0,25)
            b = random.randint(0,25)
            child = parentkey[:]
            # swap two characters in the child
            child[a],child[b] = child[b],child[a]
            deciphered = SimpleSub(child).decipher(ctext)
            score = fitness.score(deciphered)
            # if the child was better, replace the parent with it
            if score > parentscore:
                parentscore = score
                parentkey = child[:]
                count = 0
            count = count+1
        # keep track of best score seen so far
        if parentscore>maxscore:
            maxscore,maxkey = parentscore,parentkey[:]
            print ('\nbest score so far: ', maxscore, 'on iteration ', i)
            ss = SimpleSub(maxkey)
            print ('    best key: '+''.join(maxkey))
            print ('    plaintext: '+ss.decipher(ctext))
    else:
        break
```

Τα βασικά σημεία του κώδικα είναι τα ακόλουθα:

- Το input κρατείται στη μεταβλητή ctext ως string, έχοντας αφαιρέσει ωστόσο όσους χαρακτήρες δεν είναι γράμματα.
- Ξεκινάμε με ένα τυχαίο κλειδί parentkey και ένα πολύ μικρό (μεγάλο αρνητικό) maxscore. Αποκρυπτογραφούμε το κείμενο με βάση το κλειδί και υπολογίζουμε το σκορ με τη fitness

function, ως parentscore. Σημειώνουμε ότι η fitness function βρίσκεται στο αρχείο ngram_score.py Αν το τρέχον σκορ δηλαδή το parentscore είναι καλύτερο από το maxscore, προχωράμε στο επόμενο βήμα, αλλιώς σταματάμε. Δίνουμε τον κώδικα:

```
while 1:
    i = i+1
    random.shuffle(parentkey)
    deciphered = SimpleSub(parentkey).decipher(ctext)
    parentscore = fitness.score(deciphered)
    if (parentscore > maxscore):
            …

else:
  break
```

- Έχοντας ένα κλειδί και ένα αποκρυπτογραφημένο κείμενο προσπαθούμε να το βελτιώσουμε αλλάζοντάς τις θέσεις των γραμμάτων. Συγκεκριμένα, αν αλλάζοντας δύο γράμματα το νέο κρυπτοκείμενο είναι «καλύτερο» με βάση τη fitness function, τότε κρατάμε το νέο κλειδί σαν καλύτερο κλειδί, και το αποτέλεσμα της fitness function ως καλύτερο σκορ. Επαναλαμβάνουμε αυτή τη διαδικασία χίλιες φορές. Τα παραπάνω γίνονται στο ακόλουθο κομμάτι κώδικα:

```
while count < 1000:
        a = random.randint(0,25)
        b = random.randint(0,25)
        child = parentkey[:]
        # swap two characters in the child
        child[a],child[b] = child[b],child[a]
        deciphered = SimpleSub(child).decipher(ctext)
        score = fitness.score(deciphered)
        # if the child was better, replace the parent with it
        if score > parentscore:
            parentscore = score
            parentkey = child[:]
            count = 0
        count = count+1

# keep track of best score seen so far
    if parentscore>maxscore:
        maxscore,maxkey = parentscore,parentkey[:]
        print ('\nbest score so far: ', maxscore, 'on iteration ', i)
        ss = SimpleSub(maxkey)
        print ('    best key: '+''.join(maxkey))
        print ('    plaintext: '+ss.decipher(ctext))
```

- Ερχόμενοι πάλι στο εξωτερικό loop, κάνουμε μια νέα τυχαία μετάθεση των γραμμάτων φτιάχνοντας ένα νέο κλειδί και αποκρυπτογραφούμε το κείμενο. Αν το νέο σκορ είναι καλύτερο από το τρέχον maxscore ξεκινάμε εκ νέου τις χίλιες επαναλήψεις για μεταθέσεις και νέα κλειδιά, αλλιώς σταματάμε.

<u>ngram_score.py:</u>
Παραθέτουμε τον κώδικα:

```
from math import log10

class ngram_score(object):
    def __init__(self, ngramfile, sep=' '):
        ''' load a file containing ngrams and counts, calculate log probabilities '''
        self.ngrams = {}
        with open(ngramfile, 'r') as file:
            for line in file:
                key, count = line.strip().split(sep)
                self.ngrams[key] = int(count)
        self.L = len(key)
```

```python
        self.N = sum(self.ngrams.values())
        # calculate log probabilities
        for key in self.ngrams.keys():
            self.ngrams[key] = log10(float(self.ngrams[key]) / self.N)
        self.floor = log10(0.01 / self.N)

    def score(self, text):
        ''' compute the score of text '''
        score = 0
        ngrams = self.ngrams.__getitem__
        for i in range(len(text) - self.L + 1):
            if text[i:i + self.L] in self.ngrams:
                score += ngrams(text[i:i + self.L])
            else:
                score += self.floor
        return score

# Example usage:
# ngram_scorer = ngram_score('english_quadgrams.txt')
# text_to_score = "Your text here"
# score = ngram_scorer.score(text_to_score)
# print("Score:", score)
```

Τα βασικά σημεία του κώδικα είναι τα ακόλουθα:

- Η κλάση ngram_score υπολογίζει λογαριθμικές πιθανότητες n-γραμμάτων σε ένα κείμενο με βάση ένα αρχείο με n-γράμματα. Εμείς της δίνουμε ως είσοδο το αρχείο “english_quadgrams.txt” που περιέχει 4-γραμματα αγγλικών με συχνότητες εμφάνισής των. Αφού πάρει το αρχείο, η κλάση υπολογίζει τις πιθανότητες.
- Η μέθοδος score υπολογίζει το σκορ του κειμένου με βάση τις πιθανότητες των 4-γραμμάτων που περιέχει.

Το output όταν εκτελούμε το ask1_auto.py είναι το ακόλουθο:

```
Substitution Cipher solver, you may have to wait several iterations
for the correct result.

best score so far:  -6240.388453280439 on iteration  1
    best key: WAHYUPJVCEMLBDQIFRGKNXZTOS
    plaintext: THECOMPUTABLENUMBERSMAYBEDESCRIBEDBRIEFLYASTHEREALNUMBERSWHOSEEXPRESSIONSASADECIMALARECALCULABLEBYFINITEMEANSALTHOUGHTHESUBJECTOFTHISPAPERISOSTENSIBLYTHECOMPUTABLENUMBERSITISALMOSTEQUALLYEASYTODEFINEANDINVESTIGATE
COMPUTABLEFUNCTIONSOFANINTEGRALVARIABLEORAREALORCOMPUTABLEVARIABLECOMPUTABLEPREDICATESANDSOFORTHTHEFUNDAMENTALPROBLEMSINVOLVEDAREHOWEVERTHESAMEINEACHCASEANDIHAVECHOSENTHECOMPUTABLENUMBERSFOREXPLICITTREATMENTASINVOLVINGTHELEASTCU
MBROUSTECHNIQUEIHOPESHORTLYTOGIVEANACCOUNTOFTHERELATIONSOFTHECOMPUTABLENUMBERSFUNCTIONSANDSOFORTHTOONEANOTHERTHISWILLINCLUDEADEVELOPMENTOFTHETHEORYOFFUNCTIONSOFAREALVARIABLEEXPRESSEDINTERMSOFCOMPUTABLENUMBERSACCORDINGTOMYDEFINIT
IONANUMBERISCOMPUTABLEIFITSDECIMALCANBEWRITTENDOWNBYAMACHINEIGIVESOMEARGUMENTSWITHTHEINTENTIONOFSHOWINGTHATTHECOMPUTABLENUMBERSINCLUDEALLNUMBERSWHICHCOULDNATURALLYBEREGARDEDASCOMPUTABLEINPARTICULARISHOWTHATCERTAINLARGECLASSESOFN
UMBERSARECOMPUTABLETHEYINCLUDEFORINSTANCETHEREALPARTSOFALLALGEBRAICNUMBERSTHEREALPARTSOFTHEKEROSOFTHEBESSELFUNCTIONSTHENUMBERSPIEETCTHECOMPUTABLENUMBERSDONOTHOWEVERINCLUDEALLDEFINABLENUMBERSANDANEXAMPLEISGIVENOFADEFINABLENUMBERW
HICHISNOTCOMPUTABLEALTHOUGHTHECLASSOFCOMPUTABLENUMBERSISSOGREATANDINMANYWAYSSIMILARTOTHECLASSOFREALNUMBERSITISNEVERTHELESSENUMERABLEIEXAMINECERTAINARGUMENTSWHICHWOULDSEEMTOPROVETHECONTRARYBYTHECORRECTAPPLICATIONOFONEOFTHESEARGUM
ENTSCONCLUSIONSAREREACHEDWHICHARESUPERFICIALLYSIMILARTOTHOSEOFGODELTHESERESULTSHAVEVALUABLEAPPLICATIONSINPARTICULARITISSHOWNTHATTHEHILBERTIANENTSCHEIDUNGSPROBLEMCANHAVENOSOLUTION
```

Πηγή: *http://practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-simple-substitution-cipher/*

ΑΣΚΗΣΗ 2

(1).

Το $affine\ cipher$ είναι $c = Enc\big((a,b),m\big) = (ax+b) mod\ 26$.

Θεωρούμε δύο τυχαία μηνύματα m_1, m_2 και τις αποκρυπτογραφήσεις αυτών c_1, c_2 αντίστοιχα.

Είναι $c_1 \equiv (am_1 + b)\ mod\ 26$ και $c_2 \equiv (am_2 + b)\ mod\ 26$. Συνεπώς,

$$c_1 - c_2 \equiv a\ (m_1 - m_2)\ mod\ 26$$

από όπου προκύπτει ότι

$$\alpha \equiv (c_1 - c_2)(m_1 - m_2\ )^{-1}\ mod\ 26\ \ (1)$$

Για να έχει πολλαπλασιαστικό αντίστροφο $mod\ 26$ ο $m_1 - m_2$, πρέπει $\gcd(m_1 - m_2, 26) = 1$

Ο αντίπαλος λοιπόν θα επιλέξει δύο m_1, m_2 τέτοια ώστε $\gcd(m_1 - m_2, 26) = 1$. Με τον εκτεταμένο αλγόριθμο του Ευκλείδη θα βρει το $(m_1 - m_2\ )^{-1}$ και αφού ξέρει και τα c_1, c_2 υπολογίζει το α μέσω της (1). Γνωρίζοντας το α, από τον ορισμό του $affine$ βρίσκει εύκολα και το b καθώς ξέρει ένα c και ένα m. Συνεπώς, έσπασε το κρυπτοσύστημα.

(2).
Είναι

$$Enc(k,m) = Enc\big(k_2, Enc(k_1,m)\big) = (a_2(a_1 m + b_1) + b_2)(mod\ 26) = a_2 a_1 m + b_1 a_2 + b_2\ (mod\ 26)$$

το οποίο είναι ισοδύναμο με το προηγούμενο απλώς με $a = a_2 a_1$ και $b = b_1 a_2 + b_2$. Συνεπώς δεν είναι πιο ασφαλές καθώς αντιμετωπίζεται με τον ίδιο τρόπο.
Και στις δύο περιπτώσεις το πλήθος των κλειδιών είναι 26^2 δηλαδή όλοι οι συνδυασμοί των a, b για $a, b \in \{1, \dots, 26\}$. Συνεπώς και στις δύο περιπτώσεις το κρυπτοσύστημα σπάει εύκολο με εξαντλητική αναζήτηση.

ΑΣΚΗΣΗ 3

Παραθέτουμε τον κώδικα που αναπτύξαμε για την αποκρυπτογράφηση κειμένων με $Vigenere$, στη γλώσσα Python:

```
import math
import sys

letter_mappings = { 'A': 1,'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7, 'H': 8,
                    'I': 9, 'J': 10, 'K': 11, 'L': 12, 'M': 13, 'N': 14, 'O': 15, 'P': 
16,
                    'Q': 17, 'R': 18, 'S': 19, 'T': 20, 'U': 21, 'V': 22, 'W': 23, 'X': 
24,
                    'Y': 25, 'Z': 26 }

# split input_text to groups (list of lists) based on key. For example if key = 3 will 
have 3 groups and in the first group will be char 1, 4, 7, etc
def split(key, input_text):
    n = len(input_text)
    groups = [[] for _ in range(key)]
    for i in range (0, n):
        groups[i % key].append(input_text[i])
    return groups

# returns the index_of_coincidence of a text, given as a list of characters (text_list)
def index_of_coincidence(text_list, n):
    letter_freqs = {}

    for letter in text_list:
        if letter in letter_freqs:
            letter_freqs[letter] += 1
        else:
            letter_freqs[letter] = 1

    ic = 0
    for letter in letter_freqs:
        ic += (letter_freqs[letter] / n) * ( (letter_freqs[letter] - 1) / (n - 1) )

    return ic

# takes a list of decrypted character which is each group that was splitted, and de-
crypts it because each group has been encrypted with caesar
# Returns the shift which corresponds to a letter of the key.
def caesar_decrypt(group):
    letter_freqs = {}
    english_freqs =    {'A': 8.5,'B': 2.07, 'C': 4.54, 'D': 3.38, 'E': 11.16, 'F': 
1.81, 'G': 2.47, 'H': 3,
```

```
                               'I': 7.54, 'J': 0.2, 'K': 1.1, 'L': 5.49, 'M': 3.01, 'N': 6.65,
'O': 7.16, 'P': 3.17,
                               'Q': 0.2, 'R': 7.58, 'S': 5.74, 'T': 6.95, 'U': 3.63, 'V':
1.01, 'W': 1.29, 'X': 0.29,
                               'Y': 1.78, 'Z': 0.27}

    for letter in group:
        if letter in letter_freqs:
            letter_freqs[letter] += 1
        else:
            letter_freqs[letter] = 1

    def shift(letter_freqs, n):
        shifted_freqs = {}
        for letter, frequency in letter_freqs.items():
            # Convert the letter to uppercase and calculate the shifted position
            shifted_letter = chr(((ord(letter) - ord('A') + n) % 26) + ord('A'))

            # Update the frequency in the shifted dictionary
            shifted_freqs[shifted_letter] = frequency

        return shifted_freqs

    min_entropy = 2**10
    n = 0
    for i in range (1, 26):
        shifted_by_n_freqs = shift(letter_freqs, i)
        current_entropy = 0
        for letter in shifted_by_n_freqs:
            current_entropy += (shifted_by_n_freqs[letter] / len(group)) *
math.log10(english_freqs[letter])
        current_entropy *= -1
        if current_entropy < min_entropy:
            min_entropy = current_entropy
            n = i

    c = group[0]
    p = chr(((ord(group[0]) - ord('A') + n) % 26) + ord('A'))

    key_char_map = (letter_mappings[c] - letter_mappings[p]) % 26
    key_char = chr(ord('A') + key_char_map)

    return key_char

# receives an input_file and does the decryption. It outputs 5 possible plaintexts with
keys and ics
def vigenere_decrypt(input_file):
    # whole input text. with commas etc
    input_text = []

    # input text but only letters
    input_text_letters = []

    try:
        file = open(input_file, 'r')
        input_text = list(file.read())
        # list of letters of input text
        input_text_letters = [char for char in input_text if char.isalpha()]
    finally:
        file.close()

    key = 2
    # length of input but only characters
    n = len(input_text_letters)
    groups = []
    ic_counter = 0
    while ic_counter < 10:
```

```
        decrypted_text = []
        mean_ic = 0

        # split text in groups based on candidate key length
        groups = split(key, input_text_letters)
        for group in groups:
            if len(group) > 1:
                mean_ic += index_of_coincidence(group, len(group))
        # find mean index of coincidence of all groups
        mean_ic /= key

        if mean_ic >= 0.06 and mean_ic <= 0.07:
            vigenere_key = ""
            # decrypt each group seperately with caesar and find key, cause each group indicates a key character
            for group in groups:
                vigenere_key += caesar_decrypt(group)

            key_counter = 0
            # decrypt the text with key
            for i in range (0, len(input_text)):
                if key_counter == len(vigenere_key):
                    key_counter = 0
                if input_text[i].isalpha():
                    decrypted_text.append(chr ((letter_mappings[input_text[i]] - letter_mappings[vigenere_key[key_counter]]) % 26 + ord('A')))
                    key_counter += 1
                else:
                    decrypted_text.append(input_text[i])

            print()
            decrypted_text_str = ''.join(decrypted_text)
            print(vigenere_key + '\n\n' +  decrypted_text_str + '\n')
            print(mean_ic)
            print("-------------------------------------------------------------------------------------------")
            ic_counter += 1

        key += 1
        if (key == n):
            break

input_file = sys.argv[1]
vigenere_decrypt(input_file)
```

Ο παραπάνω κώδικας λειτουργεί ως εξής:

Η συνάρτηση vigenere_decrypt δέχεται ως είσοδο το αρχείο κειμένου το οποίο θέλουμε να αποκρυπτογραφήσουμε. Ξεκινώντας με μήκος κλειδιού 2, και όσο ο μετρητής ic_counter είναι μικρότερος από 10 (επειδή θέλουμε 10 πιθανά plaintexts ως έξοδο) γίνονται τα ακόλουθα σε κάθε επανάληψη:

- Η συνάρτηση split παίρνει ως είσοδο το μήκος του κλειδιού και μία λίστα με τα γράμματα του input. Επιστρέφει την λίστα από λίστες groups, η οποία χωρίζει το κείμενο σε λίστες με βάση το μήκος του κλειδιού. Πχ για μήκος κλειδιού ίσο με 3, ο πρώτος χαρακτήρας μπαίνει στην $1^η$ λίστα, ο δεύτερος στη $2^η$, ο τρίτος στην $3^η$, ο τέταρτος ξανά στην $1^η$ κοκ, δηλαδή η διαδικασία επαναλαμβάνεται κυκλικά.
- Για κάθε λίστα group από τη λίστα groups, υπολογίζουμε το δείκτη σύμπτωσης μέσω της συνάρτησης index_of_coincidence και τελικά υπολογίζουμε το μέσο δείκτη σύμπτωσης όλων των groups, διαιρώντας με το μήκος του κλειδιού. Αυτό γίνεται στο παρακάτω κομμάτι κώδικα:

```
# split text in groups based on candidate key length
        groups = split(key, input_text_letters)
```

```python
for group in groups:
    if len(group) > 1:
        mean_ic += index_of_coincidence(group, len(group))
# find mean index of coincidence of all groups
mean_ic /= key
```

- Αν ο μέσος δείκτης σύμπτωσης είναι μεταξύ των τιμών 0.06 και 0.07 τότε θεωρούμε ότι το μήκος κλειδιού που έχουμε μέχρι στιγμής ως πιθανή λύση. Αυτό, διότι κάθε group, στα αλήθεια αντιστοιχεί σε γράμματα που έχουν κρυπτογραφηθεί με το ίδιο γράμμα, δηλαδή δεν είναι παρά αποτελέσματα caesar cipher για κάποιο shift. Ο δείκτης σύμπτωσης για κείμενα που έχουν κρυπτογραφηθεί με caesar είναι κοντά στο 0.065, ωστόσο εμείς αυξάνουμε λίγο τα περιθώρια από 0.06 έως 0.07. Έχοντας λοιπόν δείκτη σύμπτωσης από 0.06 έως 0.07, εκτελούμε σε κάθε group τη συνάρτηση caesar_decrypt η οποία παίρνει ως είσοδο μία λίστα από γράμματα (group), υπολογίζει το shift με βάση μια συνάρτηση εντροπίας και βάσει αυτού βρίσκει το γράμμα με το οποίο έγινε η κρυπτογράφηση για το τρέχον group. Επιστρέφει αυτό το γράμμα. Συνεπώς, η εκτέλεση της caesar_decrypt για κάθε group, μας δίνει το κλειδί.
  Αυτά, γίνονται στο ακόλουθο κομμάτι κώδικα:

```python
if mean_ic >= 0.06 and mean_ic <= 0.07:
            vigenere_key = ""
            # decrypt each group seperately with caesar and find key, cause each group indicates a key character
            for group in groups:
                vigenere_key += caesar_decrypt(group)
```

  Η συνάρτηση εντροπίας που χρησιμοποιήσαμε είναι η ακόλουθη:

  $$H_N(f_N, f) = -\sum f_N(c) log f(c)$$

  όπου $f_N(c)$ είναι η συχνότητα εμφάνισης του χαρακτήρα c στο αποκρυπτογραφημένο κείμενο και $f(c)$ η συχνότητα εμφάνισης του χαρακτήρα c στο αλφάβητο του κειμένου, δηλαδή στα Αγγλικά.
  Υπολογίζουμε το N που ελαχιστοποιεί την παραπάνω συνάρτηση, το οποίο είναι και το shift με το οποίο έγινε η κρυπτογράφηση.

- Τέλος, έχοντας βρει το κλειδί και αποθηκεύοντάς το στη μεταβλητή vigenere_key, απλώς αποκρυπτογραφούμε το κείμενο και το εμφανίζουμε στην έξοδο μαζί με το κλειδί και το μέσο δείκτη σύμπτωσης που είχαμε υπολογίσει προηγουμένως. Επαναλαμβάνουμε έως ότου βρούμε 10 πιθανά outputs, αυξάνοντας τον μετρητή ic_counter κατά ένα.

Τρέχοντας το πρόγραμμα που γράψαμε με είσοδο το κρυπτοκείμενο της εκφώνησης λαμβάνουμε 10 πιθανά plaintexts. Το «πραγματικό» plaintext, το κλειδί και ο δείκτης σύμπτωσης φαίνονται στην παρακάτω εικόνα:

```
DHKECRYPTO

WE STAND TODAY ON THE BRINK OF A REVOLUTION IN CRYPTOGRAPHY. THE DEVELOPMENT OF
CHEAP DIGITAL HARDWARE HAS FREED IT FROM THE DESIGN LIMITATIONS OF MECHANICAL
COMPUTING AND BROUGHT THE COST OF HIGH GRADE CRYPTOGRAPHIC DEVICES DOWN
TO WHERE THEY CAN BE USED IN SUCH COMMERCIAL APPLICATIONS AS REMOTE CASH
DISPENSERS AND COMPUTER TERMINALS. IN TURN, SUCH APPLICATIONS CREATE A NEED
FOR NEW TYPES OF CRYPTOGRAPHIC SYSTEMS WHICH MINIMIZE THE NECESSITY OF SECURE
KEY DISTRIBUTION CHANNELS AND SUPPLY THE EQUIVALENT OF A WRITTEN SIGNATURE. AT
THE SAME TIME, THEORETICAL DEVELOPMENTS IN INFORMATION THEORY AND COMPUTER
SCIENCE SHOW PROMISE OF PROVIDING PROVABLY SECURE CRYPTOSYSTEMS, CHANGING
THIS ANCIENT ART INTO A SCIENCE. THE DEVELOPMENT OF COMPUTER CONTROLLED
COMMUNICATION NETWORKS PROMISES EFFORTLESS AND INEXPENSIVE CONTACT BETWEEN
PEOPLE OR COMPUTERS ON OPPOSITE SIDES OF THE WORLD, REPLACING MOST MAIL
AND MANY EXCURSIONS WITH TELECOMMUNICATIONS. FOR MANY APPLICATIONS THESE
CONTACTS MUST BE MADE SECURE AGAINST BOTH EAVESDROPPING AND THE INJECTION OF
ILLEGITIMATE MESSAGES. AT PRESENT, HOWEVER, THE SOLUTION OF SECURITY PROBLEMS
LAGS WELL BEHIND OTHER AREAS OF COMMUNICATIONS TECHNOLOGY. CONTEMPORARY
CRYPTOGRAPHY IS UNABLE TO MEET THE REQUIREMENTS, IN THAT ITS USE WOULD
IMPOSE SUCH SEVERE INCONVENIENCES ON THE SYSTEM USERS, AS TO ELIMINATE MANY
OF THE BENEFITS OF TELEPROCESSING. THE BEST KNOWN CRYPTOGRAPHIC PROBLEM IS
THAT OF PRIVACY: PREVENTING THE UNAUTHORIZED EXTRACTION OF INFORMATION FROM
COMMUNICATIONS OVER AN INSECURE CHANNEL. IN ORDER TO USE CRYPTOGRAPHY TO ENSURE
PRIVACY, HOWEVER, IT IS CURRENTLY NECESSARY FOR THE COMMUNICATING PARTIES TO
SHARE A KEY WHICH IS KNOWN TO NO ONE ELSE. THIS IS DONE BY SENDING THE KEY IN
ADVANCE OVER SOME SECURE CHANNEL SUCH AS PRIVATE COURIER OR REGISTERED MAIL.
A PRIVATE CONVERSATION BETWEEN TWO PEOPLE WITH NO PRIOR ACQUAINTANCE IS A
COMMON OCCURRENCE IN BUSINESS, HOWEVER, AND IT IS UNREALISTIC TO EXPECT INITIAL
BUSINESS CONTACTS TO BE POSTPONED LONG ENOUGH FOR KEYS TO BE TRANSMITTED
BY SOME PHYSICAL MEANS. THE COST AND DELAY IMPOSED BY THIS KEY DISTRIBUTION
PROBLEM IS A MAJOR BARRIER TO THE TRANSFER OF BUSINESS COMMUNICATIONS TO LARGE
TELEPROCESSING NETWORKS.


0.06633132431168898
```

ΑΣΚΗΣΗ 4

(1).
Αρχικά, θα δείξουμε ότι σε ένα σύστημα που έχει τέλεια μυστικότητα δεν είναι αναγκαίο κάθε κλειδί να επιλέγεται με ίδια πιθανότητα. Θα το δείξουμε με ένα αντιπαράδειγμα. Έστω ότι έχουμε το εξής κρυπτοσύστημα με χώρο μηνυμάτων $M = \{0, 1\}$ 1-bit, και χώρο κλειδιών 2-bit $K = \{00, 01, 10, 11\}$. Η κρυπτογράφηση γίνεται ως xor του μηνύματος με το πρώτο bit του κλειδιού. Θεωρούμε επίσης ότι τα κλειδιά έχουν τις εξής πιθανότητες:

$$\Pr[K = 00] = \frac{1}{8}$$
$$\Pr[K = 01] = \frac{3}{8}$$
$$\Pr[K = 10] = \frac{2}{8}$$
$$\Pr[K = 11] = \frac{2}{8}$$

Εύκολα βλέπουμε ότι το πρώτο bit του κλειδιού ικανοποιεί uniform κατανομή καθώς

$$\Pr[first\ bit\ of\ K\ is\ 0] = \Pr[first\ bit\ of\ K\ is\ 1] = \frac{1}{2}$$

Συνεπώς, το κρυπτοσύστημα έχει τέλεια μυστικότητα, χωρίς απαραίτητα τα κλειδιά να επιλέγονται με ίδια πιθανότητα.

Για το δεύτερο υποερώτημα θα δείξουμε ότι αναγκαία συνθήκη για τέλεια μυστικότητα είναι να ισχύει για τους χώρους των μηνυμάτων, κρυπτοκειμένων και κλειδιών:

$$|M| \leq |C| \leq |K|$$

- Είναι προφανώς από απαίτηση για 1-1 κρυπτογράφηση $|M| \leq |C|$
- Έστω $|C| > |K|$. Τότε $\forall x \in M, \exists\, y \in C$ τέτοιο ώστε $\Pr[C = y \mid M = x] = 0 \neq \Pr[C = y]$

(2)

(i) Είναι από τον τύπο του $Bayes$:

$$\Pr[C = y] = \frac{\Pr[M = x \mid C = y]\Pr[C = y]}{\Pr[M = x]}$$

ή $\Pr[M = x] = \Pr[M = x \mid C = y]$ άρα είναι ισοδύναμη με τη συνθήκη τέλειας μυστικότητας του $Shannon$.

(ii) Θεωρούμε τυχαίο $x \in M$. Είναι από $Shannon$:

$$\forall x \in M, y \in C: \Pr[M = x \mid C = y] = \Pr[M = x] \overset{(i)}{\Leftrightarrow} \Pr[C = y] = \Pr[C = y \mid M = x]$$

Άρα και για $x_1, x_2 \in M$ είναι $\Pr[C = y] = \Pr[C = y \mid M = x_1]$ και $\Pr[C = y] = \Pr[C = y \mid M = x_2]$.
Άρα, προκύπτει ότι $\Pr[C = y \mid M = x_1] = \Pr[C = y \mid M = x_2]$.
Τελικά, η πρόταση είναι ισοδύναμη με τη συνθήκη τέλειας μυστικότητας $Shannon$.

ΑΣΚΗΣΗ 5

Για να υπάρχει τέλεια μυστικότητα πρέπει να ισχύει $|M| \leq |C| \leq |K|$ όπου M, C, K οι χώροι των απλών κειμένων, κρυπτοκειμένων και κλειδιών αντίστοιχα. Στο τροποποιημένο one-time pad ισχύουν τα ακόλουθα:

- $|M| = 2^{\lambda}$
- $|K| = 2^{\lambda} - 1$, αφού εξαιρείται η συμβολοσειρά που αποτελείται από λ μηδενικά.

Είναι λοιπόν $|K| \leq |M|$ συνεπώς το τροποποιημένο one-time pad δεν είναι τέλεια ασφαλές.

ΑΣΚΗΣΗ 6

1).

Έστω c το ciphertext άρα $c = (km) mod\ p \Rightarrow km \equiv c\ (mod\ p)$. Είναι $\gcd(k, p) = 1$ αφού p πρώτος και $k < p$, άρα η εξίσωση έχει λύση ως προς m, συγκεκριμένα την

$$m = k^{-1}\ (mod\ p) \Leftrightarrow m\ mod\ p = k^{-1}c\ mod\ p \overset{m<p}{\Longrightarrow} m = k^{-1}c\ mod\ p$$

Άρα $Dec(k, c) = k^{-1}c\ mod\ p$

2).
Θα δείξουμε ότι κάθε αποκρυπτογράφηση δίνει σωστό αποτέλεσμα, δηλαδή ότι $m = Dek(k, c)$ όπου $c = (km) mod\, p$.
Είναι $Dec(k, c) = k^{-1} c\, mod\, p = (\, k^{-1}(km\, mod\, p))\, mod\, p$. Αφού $k^{-1} \in {Z_p}^*$ είναι $k^{-1} < p$ τότε θα είναι $Dec(k, c) = \big((k^{-1} mod\, p)(km\, mod\, p)\big) mod\, p = (k^{-1}km) mod\, p = m\, mod\, p = m$

3).
Θα χρησιμοποιήσουμε την ικανή και αναγκαία συνθήκη για τέλεια μυστικότητα που είδαμε στο μάθημα:

Έστω κρυπτοσύστημα με $|M| = |C| = |K|$. Το σύστημα έχει τέλεια μυστικότητα αν και μόνο αν ισχύουν τα εξής:
(1). Για κάθε $x \in M, y \in C$, υπάρχει μοναδικό $k \in K$ τέτοιο ώστε $Enc(k, x) = y$.
(2). Κάθε κλειδί επιλέγεται με την ίδια πιθανότητα, συγκεκριμένα $1/|K|$

Για το δικό μας κρυπτοσύστημα ισχύει ότι $M = K = {Z_p}^*$. Επίσης, $Enc(k, x) = (kx) mod\, p$ που σημαίνει ότι $Enc(k, x) \in {Z_p}^*$. Άρα, $M = K = C = {Z_p}^*$.
Επίσης, η ιδιότητα (2) ισχύει προφανώς. Θα δείξουμε ότι ισχύει και η ιδιότητα (1).

Έστω $x \in M, y \in C$. Θα δείξουμε ότι υπάρχει μοναδικό $k_1 \in K$ τέτοιο ώστε $y = Enc(k_1, x)$. Έστω ότι υπάρχει $k_2 \in K$, διαφορετικό από k_1 με $y = Enc(k_2, x)$. Άρα είναι $Enc(k_1, x) = Enc(k_2, x)$ οπότε

$$(k_1 x) mod\, p = (k_2 x) mod\, p \Rightarrow k_1 x \equiv k_2 x\ (mod\, p) \Rightarrow p \mid x(k_1 - k_2) \Rightarrow p \mid x\, |k_1 - k_2|$$

Αφού p πρώτος θα είναι είτε $p \mid x$ είτε $p \mid |k_1 - k_2|$ **άτοπο**, αφού $x, |k_1 - k_2| \in {Z_p}^*$, οπότε είναι μικρότερα του p και δε γίνεται ο p να τα διαιρεί.
Τελικά το κρυπτοσύστημα είναι τέλεια ασφαλές.

ΑΣΚΗΣΗ 7

1). Έστω n σύνθετος. Τότε, θα ισχύει $n = kx$, για $k, x \in \mathrm{N}$ και $k, x\ \neq n, 1$. Είναι

$$2^n - 1 = 2^{kx} - 1 = (2^x - 1)\big(2^{x(k-1)} + 2^{x(k-1)} + \cdots + \ 2^x + 1\big)$$

άρα προκύπτει ότι ο $2^n - 1$ είναι σύνθετος, άτοπο. Άρα n πρώτος.

2).
(i).
Είναι $(2^p - 1)(mod\, p) =\ (2^p\, mod\, p - 1\, mod\, p)(mod\, p) =\ (2^p\, mod\, p - 1)(mod\, p)$ (1)
Είναι p πρώτος περιττός, άρα $p \neq 2$, συνεπώς $p \nmid 2$ οπότε από μικρό θεώρημα $Fermat$ είναι:

$2^{p-1} \equiv 1(mod\, p) \Rightarrow\ 2^p \equiv 2(mod\, p) \Rightarrow 2^p\, mod\, p = 2\, mod\, p = 2$, αφού $p > 2$ πρώτος.

Άρα, από την (1) έχουμε $(2^p - 1)\ (mod\, p) = (2 - 1)\ (mod\, p) = 1\, mod\, p$, δηλαδή

$$M_p = 2^p - 1 \equiv 1 \quad (mod\, p)$$

(ii).
Είναι $M_p = 2^p - 1 \equiv 1\ (mod\, p) \Rightarrow p \,\big|\, \big(M_p - 1\big) \Rightarrow\ p \,\big|(2^p - 2)$, άρα

$$(2^p - 2) = kp \Rightarrow 2^p - 1 = kp + 1 \Rightarrow \ M_p = kp + 1$$

- Αν M_p πρώτος τότε $\varphi(M_p) = kp + 1 - 1 = kp$ άρα $p \mid \varphi(M_p)$
- Αν M_p σύνθετος τότε:

Έστω q πρώτος με $q \mid (M_p)$. Άρα $q \mid 2^p - 1$ και $2^p \equiv 1\ (mod\ q)$. Θεωρούμε επίσης την πολλαπλασιαστική ομάδα $Z_q{}^*$. Προφανώς $2 \in Z_q{}^*$. Έστω d η τάξη του 2 άρα $2^d \equiv 1\ (mod\ q)$. Προφανώς, πρέπει να είναι $d \mid p$ άρα αφού p πρώτος, αναγκαστικά $d = p$ ή $d = 1$. Αν $d = 1$ τότε $q \mid 2$ $- 1 = 1$ άτοπο, αφού $q > 1$. Άρα $d = p$. Η τάξη της $Z_q{}^*$ είναι $q - 1$ άρα $p \mid q - 1$. Επίσης, $q - 1$ άρτιος, συνεπώς $2 \mid q - 1$ άρα $2p \mid q - 1$ οπότε για κάποιο x φυσικό, θα είναι $q - 1 = 2px$. Δείξαμε λοιπόν ότι αν υπάρχει πρώτος q με $q \mid (M_p)$ τότε θα είναι $q - 1 = 2px$.
Ξέρουμε ότι M_p σύνθετος άρα γράφεται ως γινόμενο πρώτων παραγόντων, συνεπώς υπάρχει κάποιος πρώτος q τέτοιος ώστε $M_p = q^k l$ όπου l το υπόλοιπο ανάπτυγμα. Είναι

$$\varphi(M_p) = \varphi(q^k)\varphi(l) = q^k\left(1 - \frac{1}{q}\right)\varphi(l) = q^k\left(\frac{q-1}{q}\right)\varphi(l) = q^{k-1}(q-1)\varphi(l) = q^{k-1}2px\varphi(l)$$

Από όπου φαίνεται ότι $p \mid \varphi(M_p)$

ΑΣΚΗΣΗ 8

Αφού p, q διαφορετικοί πρώτοι, είναι $p \nmid q$ και $q \nmid p$ συνεπώς από μικρό θεώρημα του $Fermat$ θα είναι $p^{q-1} \equiv 1\ mod\ q$ και $q^{p-1} \equiv 1\ mod\ p$. Άρα $q \mid p^{q-1} - 1$ και $p \mid q^{q-1} - 1$. Επίσης αφού $q \mid q^{p-1}$ και $p \mid p^{q-1}$ προκύπτει ότι $q \mid p^{q-1} - 1 + q^{p-1}$ και $p \mid q^{q-1} - 1 + p^{q-1}$.
Συνεπώς για κάποιο x θα είναι $xq = q^{q-1} - 1 + p^{q-1}$ και $p \mid xq$. Αφού p, q πρώτοι τότε αναγκαστικά $p \mid x$ άρα για κάποιο y θα είναι $py = x$ και αφού $xc = q^{q-1} - 1 + p^{q-1}$ τότε
$pqy = q^{q-1} - 1 + p^{q-1}$ ή $pq \mid q^{q-1} - 1 + p^{q-1}$ ή $q^{q-1} + p^{q-1} \equiv 1\ mod\ pq$

ΑΣΚΗΣΗ 9

Είναι

$$\sum\nolimits_{\beta \in Zp^*} \beta = 1 + 2 + \cdots + (p-1) = \sum\nolimits_{k=1}^{p-1} k = \frac{(p-1)p}{2}$$

Αφού p πρώτος είναι και περιττός άρα $p - 1$ άρτιος και $\frac{p-1}{2} = a \in N - \{1\}, αφού\ p > 2$, οπότε

$$\sum\nolimits_{\beta \in Zp^*} \beta = \alpha p \equiv 0\ (mod\ p)$$

Θεωρούμε την πολλαπλασιαστική ομάδα $Z_p{}^*$. Κάθε στοιχείο $\beta \in Zp^*$ έχει μοναδικό αντίστροφο $\beta^{-1} \in Zp^*$. Άρα, τα στοιχεία $\{1, 2, \dots, p-1\}$ της ομάδας απεικονίζονται με μοναδικό τρόπο σε μια μετάθεση των ίδιων στοιχείων. Έτσι

$$\sum\nolimits_{k=1}^{p-1} k = \sum\nolimits_{k=1}^{p-1} k^{-1} = \frac{(p-1)p}{2}$$

Άρα

$$\sum\nolimits_{\beta \in Zp^*} \beta^{-1} = \frac{(p-1)p}{2}$$

και ομοίως με πριν καταλήγουμε ότι

$$\sum\nolimits_{\beta \in Zp^*} \beta^{-1} \equiv 0 \ (mod\ p)$$

ΑΣΚΗΣΗ 10

1). Ευθύ: Είναι n πρώτος και αφού $m < n$ θα είναι

$$\sum_{j=1}^{m} \gcd(n,j) = \sum_{j=1}^{m} \gcd(n,j) = m$$

αφού για κάθε $1 \leq j \leq m < n$ είναι $\gcd(n,j) = 1$ με n πρώτο.

Αντίστροφο: Για να είναι

$$\sum_{j=1}^{m} \gcd(n,j) = m$$

πρέπει για κάθε j να είναι $\gcd(n,j) = 1$. Αυτό, γιατί $\gcd(n,j) \in N^+$ οπότε αφού έχουμε m όρους, για να αθροίζουνε σε m, πρέπει αναγκαστικά κάθε όρος να ισούται με τη μονάδα. Άρα για κάθε $j \in \left\{1, \dots, \sqrt{\lfloor n \rfloor}\right\}$ είναι $\gcd(n,j) = 1$.

Πρέπει να δείξουμε ότι και για κάθε $j \in \left\{\sqrt{\lfloor n \rfloor} + 1, \dots, n\right\}$ είναι $\gcd(n,j) = 1$. Έστω ότι υπάρχει j με $\gcd(n,j) = l > 1$ και $j \geq \sqrt{\lfloor n \rfloor} + 1$. Άρα $n = jl$ με $l \leq \lfloor n \rfloor$, οπότε $n \mid l$ άτοπο.

Τελικά, για κάθε $j \in \{1, \dots, n\}$ είναι $\gcd(n,j) = 1$ συνεπώς n πρώτος.

2). Έχουμε τον εξής αλγόριθμο:

Αν n ≤ 3:
    Τύπωσε «n πρώτος»
    ΤΕΛΟΣ
    Υπολόγισε $\lfloor \sqrt{n} \rfloor$
    $m \leftarrow \lfloor \sqrt{n} \rfloor$
    Για j από 1 έως m:
        Υπολόγισε gcd(n, j)
        Αν gcd(n, j) ≠ 1
            Τύπωσε «n σύνθετος»
            ΤΕΛΟΣ
        Αλλιώς, συνέχισε την επανάληψη
    Τύπωσε «n πρώτος»

Η πολυπλοκότητα του αλγορίθμου ελέγχου πρώτων είναι $O(\lfloor\sqrt{n}\rfloor log\, n) = O(logn\sqrt{n})$ υποθέτοντας ότι χρησιμοποιούμε τον αλγόριθμο του Ευκλείδη για το gcd με πολυπλοκότητα $O(logn)$. Η πολυπλοκότητα του αλγορίθμου $Miller\ Rabin$ είναι $O(klog^3(n))$ όπου k ο αριθμός των επαναλήψεων. Η πολυπλοκότητα του αλγορίθμου που βρίσκει όλους τους διαιρέτες έως $\lfloor\sqrt{n}\rfloor$ είναι $O(\lfloor\sqrt{n}\rfloor) = O(\sqrt{n})$. Συγκεκριμένα, χρησιμοποιήσαμε τον ακόλουθο αλγόριθμο:

```
Υπολόγισε ⌊√n⌋
m ← ⌊√n⌋
res = [ ]
Για j από 1 έως m:
        Υπολόγισε n mod j
        Αν n mod j = 0:
                res. append(j)            // j διαιρεί το n
Τύπωσε res
```

ΑΣΚΗΣΗ 11

Βλέπουμε ότι το L_n περιέχει στοιχεία τα οποία ανήκουν στο ${Z_n}^+$. Επίσης, η ${Z_n}^*$ περιέχει όλα τα στοιχεία του ${Z_n}^+$. Αρκεί λοιπόν να δείξουμε ότι όλα τα στοιχεία του ${Z_n}^*$ ανήκουν στο L_n. Έστω στοιχείο $a \in {Z_n}^*$. Θα δείξουμε ότι το στοιχείο αυτό ανήκει και στο L_n.

Αρχικά, αφού n πρώτος και αφού $a \in {Z_n}^*$ που σημαίνει ότι $n \nmid a$, τότε από μικρό θεώρημα $Fermat,$ θα είναι $\alpha^{n-1} = 1\ (mod\ n)$. Επίσης, είναι $n-1$ άρτιος συνεπώς γράφεται ως $t2^h$ για t περιττό. Άρα, είναι $\alpha^{n-1} = \alpha^{t2^h} = 1\ (mod\ n)$.

Για κάθε k στο $\{0, \dots, h-1\}$ ισχύει ότι:

$$\alpha^{t2^{k+1}} = 1 \Rightarrow \left(\alpha^{t2^k}\right)^2 = 1 \Rightarrow \alpha^{t2^k} = \pm 1\ (mod\ n)$$

Το τελευταίο ισχύει καθώς η εξίσωση $x^2 = 1\ (mod\ n)$ έχει μοναδικές λύσεις $x = \pm 1\ (mod\ n)$ για n πρώτο.

<u>Απόδειξη:</u>

$$x^2 = 1\ (mod\ n) \Rightarrow n \mid x^2 - 1 \Rightarrow n \mid (x-1)(x+1)$$

Από γνωστή ιδιότητα, αφού n πρώτος, τότε είτε $n \mid (x-1)$ είτε $n \mid (x+1)$. Άρα, $x = \pm 1\ (mod\ n)$

■

Τελικά, δείξαμε ότι $L_n = {Z_n}^*$

ΑΣΚΗΣΗ 12

Εφόσον τα στοιχεία της B_1 είναι $\alpha_1 \in G_1$ θα ανήκουν στη G_1 άρα $B_1 \subseteq G_1$.
Μένει να δείξουμε ότι
$(B_1, +_1)$ είναι ομάδα.

- ➔ Έστω $\alpha_i, \alpha_j \in B_1$. Από ορισμό B_1 θα είναι $(\alpha_i, b_{1i}) \in B$ και $(\alpha_j, b_{1j}) \in B$, με $b_{1i}, b_{1j} \in G_2$.
  Αφού B ομάδα, τότε λόγω κλειστότητας $(\alpha_i, b_{1i}) + (\alpha_j, b_{1j}) \in B$ άρα
  $(\alpha_i +_1 \alpha_j,\ b_{1i} +_2 b_{1j}) \in B$. Είναι $b_{1i} +_2 b_{1j} \in G_2$ λόγω κλειστότητας ομοίως και

$\alpha_i +_1 \alpha_j \in G_1$ άρα από ορισμό B_1 βλέπουμε ότι $\alpha_i +_1 \alpha_j \in B_1$. Άρα, δείξαμε την κλειστότητα.

- Η προσεταιριστικότητα είναι προφανής, αφού G_1 ομάδα και $B_1 \subseteq G_1$.

- Θεωρούμε τα ουδέτερα στοιχεία $e_1 \in G_1$ και $e_2 \in G_2$. Προφανώς, θα είναι για $a_1 \in G_1 : a_1 +_1 e_1 = a_1$ και για $a_2 \in G_2 : a_2 +_2 e_2 = a_2$.

  Είναι $(a_1, b_1) + (e_1, e_2) = (a_1, b_1)$ άρα το (e_1, e_2) είναι ουδέτερο για την $G_1 \, x \, G_2$ και για τη B. Συνεπώς, αφού $e_2 \in G_2$, από ορισμό B_1 βλέπουμε ότι $e_1 \in B_1$ άρα και η B_1 έχει ουδέτερο.

- Θεωρούμε στοιχείο $a_1 \in B$. Τότε υπάρχει και $b_1 \in G_2$ τέτοιο ώστε$(a_1, b_1) \in B$. Είναι $(e_1, e_2) \in B$ άρα για $(a_1, b_1) \in B$ υπάρχει $(a_2, b_2) \in B$, δηλαδή αντίστροφο τέτοιο ώστε $(a_1, b_1) + (a_2, b_2) = (e_1, e_2) \Rightarrow \begin{cases} \alpha_1 +_1 \alpha_2 = e_1 \\ b_1 +_2 b_2 = e_2 \end{cases}$. Προκύπτει λοιπόν ότι $\alpha_2 = {\alpha_1}^{-1}$ και $b_2 = {b_1}^{-1}$ άρα αφού $(a_2, b_2) \in B$ είναι $({\alpha_1}^{-1}, {b_1}^{-1}) \in B$ και αφού ${b_1}^{-1} \in G_2$ και ${\alpha_1}^{-1} \in G_1$ τότε ${\alpha_1}^{-1} \in B_1$ άρα υπάρχει και αντίστροφο.

Καταλήξαμε τελικά ότι B_1 είναι υποομάδα της G_1.

ΑΣΚΗΣΗ 13

1).

Είναι $g^{p-1} \equiv 1 (mod\, p)$ (1), αφού g γεννήτορας. Επίσης, $d \mid (p-1)$ άρα $p - 1 = kd$, για $k \in \{2, \ldots, p-2\}$. Άρα από (1) έχουμε:

$$g^{kd} = 1 \; (modp) \Rightarrow \left(g^k\right)^d = 1 \; (mod\, p)$$

Πρέπει τώρα να δείξουμε ότι το d είναι όντως η τάξη του g^k, δηλαδή δεν υπάρχει μικρότερο με την παραπάνω ιδιότητα. Έστω ότι υπάρχει $l < d$ τέτοιο ώστε $\left(g^k\right)^l = 1 \; (mod\, p)$. Τότε $g^{\frac{(p-1)d}{l}} = 1 \; (mod\, p)$. Όμως, αφού $l < d \Rightarrow {}^{l}/_{d} < 1$ άρα ${}^{(p-1)l}/_{d} < p - 1$ άρα το g δεν είναι γεννήτορας αφού η τάξη του δεν είναι $p - 1$, άτοπο.

Τελικά, το ζητούμενο b είναι $g^k \; (mod\, p)$ με g, k, p γνωστά.

2).

Η ${Z_p}^*$ περιέχει ακριβώς μια κυκλική υποομάδα τάξης d, από θεμελιώδες θεώρημα κυκλικών υποομάδων, έστω H_d. Προφανώς, ο αριθμός των στοιχείων τάξης d στη ${Z_p}^*$, ισούται με τον αριθμό γεννητόρων της H_d. Αρκεί λοιπόν να βρούμε πόσους γεννήτορες έχει η H_d.

Θα μας βοηθήσει πρώτα να χρησιμοποιήσουμε την ακόλουθη ιδιότητα, την οποία θα αποδείξουμε:

Έστω G κυκλική ομάδα με $g \in G$ γεννήτορα και $|G| = n$ και έστω $k \in N$. Τότε,

$$|g^k| = \frac{n}{\gcd(n, k)}$$

Απόδειξη:

Είναι $(g^{\gcd(n,k)})^{\frac{n}{\gcd(n,k)}} = g^n = e$ άρα

$$| g^{\gcd(n,k)} | \leq \frac{n}{\gcd(n,k)} \ (1)$$

Έστω

$$b = | g^{\gcd(n,k)} | < \frac{n}{\gcd(n,k)}$$

Τότε $e = (g^{\gcd(n,k)})^b = g^{\mathrm{b}\gcd(n,k)}$
με $b\gcd(n,k) < n$ άρα g όχι γεννήτορας, άτοπο.

■

Άρα και για την H_d είναι:
Για $k \in \{1, \dots, d\}$ είναι

$$g^k = \frac{d}{\gcd(k,d)}$$

Είναι $|g^k| = d$, δηλαδή g^k γεννήτορας αν και μόνο αν $\gcd(k,d) = 1$. Άρα, οι γεννήτορες είναι $\varphi(d)$, ομοίως και τα στοιχεία τάξης d στη $Z_p{}^*$.

(3).
Είναι $b = g^d$ άρα το b παράγει τη μοναδική κυκλική υποομάδα τάξης d,
έστω H_d. Όλα τα στοιχεία τάξης d παράγουν την H_d, δηλαδή είναι γεννήτορες. Συνεπώς, παράγουν την H_d, δηλαδή είναι γεννήτορες. Συνεπώς υπάρχουν $\varphi(d)$ γεννήτορες, αφού υπάρχουν και $\varphi(d)$ στοιχεία τάξης d.

(4).
Από θεμελιώδες θεώρημα κυκλικών υποομάδων, υπάρχει μόνο μία κυκλική υποομάδα τάξης d στο $Z_p{}^*$, συγκεκριμένα η υποομάδα $< g^{\frac{p-1}{d}} >$.

Απόδειξη:
Για $d \mid p-1$ είναι

$$| < g^{\frac{p-1}{d}} > | = \left| g^{\frac{p-1}{d}} \right| = \frac{p-1}{\gcd\left(p-1, \frac{p-1}{d}\right)} = \frac{p-1}{\frac{p-1}{d}} = d$$

Άρα η $< g^{\frac{p-1}{d}} >$ είναι κυκλική υποομάδα τάξης d. Μένει να δείξουμε ότι είναι μοναδική. Έστω $H \leq G$ με $|H| = d$ με $d \mid p-1$. Άρα H υποομάδα της G οπότε θα είναι $H =< g^m >$ για κάποιο m με $m \mid p-1$. Είναι $m \mid p-1$ επειδή αν l η τάξη του g^m τότε $(g^m)^l = e$ οπότε πρέπει $l = p-1$
Άρα

$$|H| = | < g^m > | = |g^m| = \frac{p-1}{\gcd(p-1,m)} = \frac{p-1}{m}$$

αφού $m \mid p-1$. Άρα,

$$d = \frac{p-1}{m}$$

οπότε

$$m = \frac{p-1}{d}$$

Τελικά $H = < g^{\frac{p-1}{d}} >$.

(5).
Το στοιχείο h παράγει μια κυκλική υποομάδα τάξης d έστω H_d. Είναι

$$(h^d)^t = e^t = e$$

Και $(h^d)^t = (h^t)^d = e$ για κάθε t.
Άρα τα h^t για $t \in \{1, \dots, d\}$ είναι όλα τα στοιχεία της H_d. Αν το στοιχείο a ανήκει στην H_d θα γράφεται ως $a = h^{t_i} \,(mod\, p)$, άρα θα ισχύει για αυτό ότι $a^d = (h^{t_i})^d = 1 \,(mod\, p)$. Συνεπώς αρκεί να ελέγξουμε αν $a^d = 1 \,(mod\, p)$.
Βέβαια, πρέπει να δείξουμε πως αν το a δεν ανήκει στην H_d τότε θα είναι $a^d \neq 1 (mod\, p)$.
Πράγματι, έστω $a^d = 1 \,(mod\, p)$ με a να μην ανήκει στην H_d. Τότε είτε το a έχει τάξη d είτε κάποιο k με $k \mid d$.

- Αν a έχει τάξη d, τότε $a \in H_d$ άτοπο.
- Αν a έχει τάξη k: Αφού $k \mid d$, τότε υπάρχει μοναδική κυκλική υποομάδα τάξης k της H_d, έστω H_{kd}. Άρα $a \in H_{kd}$ και αφού $H_{kd} \leq H_d$ τότε $a \in H_d$ άτοπο.

Οπότε αρκεί να ελέγξουμε αν $a^d = 1 \,(mod\, p)$, κάτι το οποίο γίνεται σε πολυωνυμικό χρόνο.

ΑΣΚΗΣΗ 14

Παραθέτουμε τον κώδικα για τον έλεγχο πρώτων αριθμών *Miller Rabin* σε γλώσσα Python:

```python
import random

def MillerRabin(k, n, r, a):
    b = pow(a, k ,n)
    if b == 1 or b == n - 1:
        return True
    for i in range(0, r - 1):
        b = pow(b, 2, n)
        if b == n - 1:
            return True
    return False

def IsPrime(base, exp, offset):
    n = base ** exp - offset
    if n == 2 or n == 3 or n == 1:
        return True

    k = n - 1
    r = 0
    while k % 2 == 0:
        k //= 2
        r += 1
    for i in range(1, 30):
        a = random.randint(2, n - 2)
        if not MillerRabin(k, n, r, a):
            print("Not Prime\n")
            return False
    print("Prime\n")
    return True

print("67280421310721 ")
IsPrime(67280421310721, 1, 0)
```

```
print("170141183460469231731687303715884105 7 ")
IsPrime(170141183460469231731687303715884105 7, 1, 0)

print("2^1001 - 1 ")
IsPrime(2, 1001, 1)

print("2^2281 - 1 ")
IsPrime(2, 2281, 1)

print("2^9941 - 1 ")
IsPrime(2, 9941, 1)

print("2^19939 - 1 ")
IsPrime(2, 19939, 1)
```

Σημειώνουμε ότι οι αριθμοί δίνονται ως είσοδος στη μορφή $βάση^{εκθέτης} - offset$.

Τα αποτελέσματα για τους ζητούμενους αριθμούς είναι τα ακόλουθα:

```
67280421310721
Prime

170141183460469231731687303715884105 7
Not Prime

2^1001 - 1
Not Prime

2^2281 - 1
Prime

2^9941 - 1
Prime

2^19939 - 1
Not Prime
```