Ψηφιακές Επικοινωνίες 1

Ονοματεπώνυμο : Νικόλας Φιλιππάτος
ΑΜ: 1072754
Εργασία: 3η

Ερωτημα

Η έξοδος μιας πηγής είναι η ακολουθία

```
1111100010101010101000110000000001010101000000001001111000010101111110000001010101100
```

Κωδικοποιείστε την έξοδο της πηγής χρησιμοποιώντας τον κώδικα Lempel-Ziv.
Εξηγείστε πότε αυτός ο κώδικας είναι αποδοτικός.

Θεωρια

Ξεκιναμε με το να σπαμε το string στους μικροτερο δυνατους συνδυασμους που δεν εχουν προκυψει ξανα (Στηλη word).

Για να κωδικοποιησουμε την καθε λεξη, ξεκιναμε και κοιταζουμε την στηλη word και τις κραταμε σε μια λιστα iterated.
Αν υπαρχει καποιος συνδυασμος απο τον οποιο ξεκιναει η λεξη που κοιταμε, βρισκουμε τον μεγαλυτερο συνδυασμο που υπαρχει στο iterated .
Για το δευτερο μερος της κωδικοποιημενης λεξης κοιταζουμε ολες τις κωδικοποιημενες λεξεις και βαζουμε με την λεξικογραφικη σειρα, στις ιδιες κωδικοποιημενες λεξεις τους αντιστοιχους 0 και 1

Για την κωδικοποιηση της καθε λεξης :
Κοιταζουμε την στηλη iterated. Αν η λεξη μας, πλυν το τελευταιο bit υπαρχει στην λιστα iterated παιρνουμε το μεγαλυτερο αριθμο που εμφανιζεται και καταγραφουμε σαν πρωτο μερος την θεση εμφανισης του αριθμου που βρηκαμε.

Οταν περασουμε ολες τις λεξεις, κοιταζουμε τις στηλες word και encoded.
Η στηλη encoded περιεχει μονο το πρωτο μερος. Για να βρουμε το δευτερο, κοιταζουμε αμα υπαρχουν διπλοτυπα.
Οταν βρουμε διπλοτυπα, αναθετουμε την τιμη 0 στο δευτερο μερος του encoded word, οπου το word εκεινης της σειρας ειναι μικροτερο, και 1 στο αλλο.

Ο κωδικας αυτος ειναι αποδοτικος οταν εχουμε μεγαλα αρχεια, πιο συγκεκριμενα οταν η κωδικη λεξη ειναι μικρότερη απο τον μεσο ορο των μεγεθων των συμβολων που βγαλαμε απο την τεμαχιση του κειμενου

Lempel-Ziv Dictionary

index	position	word	encoded
1	00001	1	000001
2	00010	11	000011
3	00011	110	000100
4	00100	0	000000
5	00101	01	001001
6	00110	010	001010
7	00111	10	000010
8	01000	101	001111
9	01001	0100	001100
10	01010	011	001011
11	01011	00	001000
12	01100	000	010110
13	01101	0000	011000
14	01110	1010	010000
15	01111	10100	011100
16	10000	00000	011010
17	10001	01001	010011
18	10010	111	000101
19	10011	00001	011011
20	10100	0101	001101
21	10101	1111	100101
22	10110	100	001110
23	10111	000010	100110
24	11000	10101	011101
25		100	001110

Code

```
from collections import defaultdict


def save_to_file(result, file):
    with open(file, "w") as f:
        for line in result:
            f.write(line)


def save_code():
    lines = ["```python\n\n"]
    with open(__file__, "r") as f:
        for line in f:
            lines.append(line)
    lines.append("```")
    return lines


def print_tables(input_string):
    dictionary, encoded_dict = lemziv_encoding(input_string)

    # max_value is used for the filling of the dictionary position
    max_value = len(dictionary.values())

    result = []
    result.append(
        f"| {'index':^10} | {'position':^10} | {'word':^10} | {'encoded':^10} |\n"
    )

    result.append(f"| {'-':^10} | {'-':^10} | {'-':^10} | {'-':^10} |\n")

    for i, (phrase, index) in enumerate(dictionary.items()):
        temp_encoded = encoded_dict[phrase]

        result.append(
            f"| {i+1:^10} | {index:^10} | {phrase:^10} | {temp_encoded:^10} |\n"
        )

    return "".join(result)


def find_max_position(input_string):
    length = len(input_string)
    counter = 0
    while length > 0:
        length -= counter**2
        counter += 1
    return counter


def create_length_dictionary(max_length):
    """Returns a dictionary with keys the length of the values and values a sorted list of binary
representations"""
    binary_dict = defaultdict(list)
    for length in range(1, max_length):
        for i in range(2**length):
            binary_dict[length].append(bin(i)[2:].zfill(length))

    return binary_dict
```

```
def length_binary(temp_word):
    """turns a number to binary and returns the length of the binary representation"""
    temp_word = bin(int(temp_word))[2:]
    return len(temp_word)


def temp2bin_filled(temp_word, max_length):
    """turns a number to binary and returns it as a string filled with zeros"""
    temp_word = bin(int(temp_word))[2:].zfill(max_length)
    return temp_word


def lempel_ziv_dict(input_string):
    dictionary = {}

    # Max length of temp_word that I can go looking at
    max_length = find_max_position(input_string)

    # Dictionary that has keys as all the possible lengths, and values lists of the binary
representations
    # of numbers iterated
    binary_dict = create_length_dictionary(max_length)

    # assign a temp_string for no reason
    temp_string = input_string

    # counting how many different temp_words will be found
    counter = 1

    # checks the temp_string if it is empty and continues
    # iterates through all the possible words of length 1 to max_length
    for _ in range(len(temp_string)):
        for length in range(1, max_length):
            temp_word = temp_string[:length]
            if temp_word not in dictionary.keys() and temp_word in binary_dict[length]:
                dictionary[temp_word] = counter
                # removes that temp_word from the string we are examining and continues the while loop
                temp_string = temp_string[length:]
                counter += 1
                break

    dictionary = {
        k: temp2bin_filled(v, length_binary(counter)) for k, v in dictionary.items()
    }

    if temp_string:
        print(f"{temp_string} : {counter}")

    return dictionary


def lemziv_encoding(input_string):
    dictionary = lempel_ziv_dict(input_string)

    max_value = len(max(dictionary.values()))

    encoded_dict = defaultdict(str)

    for word in dictionary.keys():
        # check if the word is 0 or 1
        if word == "0" or word == "1":
            # encoded_dict[word] = temp2bin_filled(0, max_value) + word
            encoded_dict[word] = temp2bin_filled(0, max_value) + word
```

```
            continue
        temp_pos = dictionary[word[:-1]]
        temp_bit = word[-1]
        encoded_dict[word] = temp_pos + temp_bit
    return dictionary, encoded_dict


def main():
    input_string =
"1111100010101010101000110000000001010101000000001001111000010101111110000001010101100"
    # input_string = "10101101001001110101000011001110101100011011"
    result = []

    result.append("## Lempel-Ziv Dictionary\n\n")
    result.append(print_tables(input_string))
    result.append("\n\n")

    print("".join(result))

    # Saving Code

    result.append("## Code\n\n")
    result.append("".join(save_code()))
    result.append("\n\n")

    save_to_file(result, "../MD_Reports/assignment-3-code-results.md")


if __name__ == "__main__":
    main()
```