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

Initial Output

index	position	word	encoded
1	00001	1	0,1
2	00010	11	1,1
3	00011	110	10,0
4	00100	0	0,0
5	00101	01	100,1
6	00110	010	101,0
7	00111	10	1,0
8	01000	101	111,
9	01001	0100	101,
10	01010	011	101,1
11	01011	00	100,0
12	01100	000	1011,0
13	01101	0000	1100,0
14	01110	1010	1000,0
15	01111	10100	1110,0
16	10000	00000	1101,0
17	10001	01001	1001,0
18	10010	111	10,1
19	10011	00001	01101,0
20	10100	0101	1111,0
21	10101	1111	10010,0
22	10110	100	00111,0
23	10111	000010	10011,0
24		10101	1110,0

Lempel-Ziv Dictionary

index	position	word	encoded
1	0001	1	000001
2	0010	11	000011

index	position	word	encoded
3	0011	110	000100
4	0100	0	000000
5	0101	01	001001
6	0110	010	001010
7	0111	10	000010
8	1000	101	001111
9	1001	0100	001100
10	1010	011	001011
11	1011	00	001000
12	1100	000	010111
13	1101	0000	011001
14	1110	1010	010001
15	1111	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	100111
24		10101	011101

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)
    # dictionary = lempel_ziv_dict(input_string)

    # max_encoded_value is used for the filling of the first part of the encoded word
    max_encoded_value = max(encoded_dict.values(), key=lambda x: x[0])[0]
    max_encoded_value = length_binary(max_encoded_value)

    # max_value is used for the filling of the dictionary position
    max_value = max(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()):
        dict_position = (
            temp2bin_filled(index, length_binary(max_value) - 1)
            if i < max_value - 1
            else " "
        )

        temp_encoded_binary_position = temp2bin_filled(
            encoded_dict[phrase][0], max_encoded_value
        )

        temp_encoded = temp_encoded_binary_position + str(encoded_dict[phrase][1])

        result.append(
            f"| {i+1:^10} | {dict_position:^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 create_grouped_dict(dictionary):
    """Returns a dictionary with keys the length of the values and values a sorted list of
dictionaries"""

    grouped_dict = defaultdict(list)
    for value, index in dictionary.items():
        temp_d = {value: index}
        temp_key = len(str(value))
        grouped_dict[temp_key].append(temp_d)
        grouped_dict[temp_key].sort(key=lambda x: list(x.keys())[0])
    return grouped_dict


def temp2bin(temp_word):
    """turns a number to binary and returns it as a string"""
    temp_word = bin(int(temp_word))[2:]
    return temp_word


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 get_key_from_value(dictionary, value):
    """Returns the key of a dictionary from a value"""
    for key, val in dictionary.items():
        if val == value:
            return key


def get_sorted_list(dictionary):
    """Returns a sorted list of the values of a dictionary"""
    sorted_list = []
    sorted_values = sorted(dictionary.values())
    for value in sorted_values:
        sorted_list.append(get_key_from_value(dictionary, value))
    return sorted_list


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


def length_word_checker(word, dictionary):
    """Checker for the same length words in the dictionary and returns the biggest one"""
    last_digit = 1
    grouped_dict = create_grouped_dict(dictionary)

    t_list = grouped_dict[len(word)]
    t_sorted_list = []
    for i in t_list:
        item = list(i.keys())[0]
        if item != word:
            t_sorted_list.append(item)

    for s_word in t_sorted_list:
        # We only want the words that start off the same
        if s_word[:-1] == word[:-1]:
            if s_word > word:
                last_digit = 0
            elif s_word < word:
                last_digit = 1

    return last_digit


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

    # encoded will be a dictionary containing as key the word to be encoded
    # and as value a list of the position of the highest closest word, and one bit
    encoded_dict = defaultdict(list)

    sorterd_list = get_sorted_list(dictionary)
    iterated = []
```

```
    for word in sorterd_list:
        temp_list = [0, 0]
        temp_list[1] = length_word_checker(word, dictionary)

        for w_iter in sorted(iterated):
            temp_list[1] = length_word_checker(word, dictionary)

            if word.startswith(w_iter):
                # get the last biggest word that was inserted and that starts with w
                temp_list[0] = dictionary[w_iter]

                # start checking for the same words in the grouped_dict
                temp_list[1] = length_word_checker(word, dictionary)

        encoded_dict[word] = temp_list
        iterated.append(word)

    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()
```