Sprawozdanie z laboratorium Teorii informacji i kompresji danych

Lab nr 4 Kompresja bezstratna - Wstęp

Lab nr 5 Kompresja bezstratna – Kodowanie Huffmana

Wojciech Szczepaniak

Index: 136808

Data zajęć: 21.01.2020 (czwartek 13:30)

1. Kodowanie binarne

W celu realizacji zadania należało przygotować następujące metody:

a. Create

```
from bitarray import bitarray
def create(counted_signs):
    sign_len = len(counted_signs)
    code_len = len(bin(sign_len)[2:])
    code_tracker = {}
    counter = 0
    for sign_tuple in counted_signs:
        tmp = bitarray(bin(counter)[2:])
        for j in range(code_len-1):
            tmp.insert(0,False)
        code_tracker[sign_tuple[0]] = tmp[-code_len:]
        counter+=1
    return(code_tracker)
```

b. Encode

```
def encode(code,text):
    encoded_text = bitarray(0)
    for i in text:
        encoded_text= encoded_text + code[i]
    return(encoded_text)
```

c. Decode

```
def decode(code,coded_text):
    decoded_text = ''
    tmp = bitarray(0)
    for i in coded_text:
        tmp = tmp + bitarray(str(int(i)))
        try:
            for key,value in code.items():
                if tmp == value:
                    decoded_text = decoded_text + str(key)
                    tmp = bitarray(0)
        except:
            pass
    return(decoded_text)
```

d. Save

```
def save(encoded_text,filename,filename_code,code):
    f = open(filename, "w")
    f.write(str(len(code.values()))+"\n")
    for key, value in code.items():
        f.write(key+" "+str(value.to01())+"\n")
    f.close()
    with open(filename_code, 'wb') as fh:
        encoded_text.tofile(fh)
    return(1)
```

e. Load

```
def load(filename_text,filename_code):
    code = {}
    f = open(filename_code, "r")
    f_list = list(f)
    num_keys = f_list[0]
    del f_list[0]
    for i in range(int(num_keys)):
        code[f_list[i].rsplit(" ",1)[0]] = bitarray(f_list[i].rsplit(" ",1)[1].replace("\n",""))
    with open(filename_text,'rb') as file:
        text = bitarray(0)
        text.fromfile(file)
    return(code,text)
```

f. Zadanie 1

```
import string
def zadanie1(filename):
    filename = 'norm_wiki_sample_cut.txt'
    filename_save = 'norm_wiki_sample_save.txt'
    filename_code = 'norm_wiki_sample_code.txt'
    znaki = list(string.ascii_lowercase)
    liczby = list(string.digits)
    all_signs = znaki +liczby+ list(" ")

    with open(filename,"r") as f1:
        data = f1.read()

    sign_tracker = {}
    for i in range(len(all_signs)):
        sign_tracker[all_signs[i]] = 0
    for i in data:
        try:
            sign_tracker[i] += 1
        except:
            #other signs
            pass

    sign_tracker = dict(sorted(sign_tracker.items(), key=lambda item: item[1], reverse = True))
    sign_list = [(k,v) for k,v in sign_tracker.items()]

    code = create(sign_list)
    encoded_text = encode(code,data[0:250])
    save(encoded_text,filename_save,filename_code,code)
    loaded_code,loaded_text = load(filename_code,filename_save)
    decoded_text = decode(loaded_code,loaded_text)
    if data[0:250] == decoded_text:
        msg = "poprawnie"
    else:
        msg = "niepoprawnie"
    return(code,msg,decoded_text,sign_list)

code,msg,decoded_text,alfabet = zadanie1('norm_wiki_sample.txt')
```

Wynik zadania pierwszego:

Wynikiem zadania pierwszego jest wygenerowany kod 6 bitowy, string “poprawnie” bądź “niepoprawnie” w zależności czy odkodowany tekst jest identyczny z tekstem początkowym oraz odkodowany tekst w tym wypadku identyczny tekst jak w pierwszych 250 znakachz pliku 'norm_wiki_sample.txt'.

```
{' ': bitarray('000000'), 'e': bitarray('000001'), 'a': bitarray('000010'), 't': bitarray('00001
1'), 'i': bitarray('000100'), 'n': bitarray('000101'), 'o': bitarray('000110'), 'r': bitarray('0001
11'), 's': bitarray('001000'), 'h': bitarray('001001'), 'l': bitarray('001010'), 'd': bitarray('001
011'), 'c': bitarray('001100'), 'm': bitarray('001101'), 'u': bitarray('001110'), 'f': bitarray('00
1111'), 'p': bitarray('010000'), 'g': bitarray('010001'), 'b': bitarray('010010'), 'w': bitarray('0
10011'), 'y': bitarray('010100'), 'v': bitarray('010101'), 'k': bitarray('010110'), '1': bitarray
('010111'), '0': bitarray('011000'), '9': bitarray('011001'), '2': bitarray('011010'), 'j': bitarra
y('011011'), '8': bitarray('011100'), '3': bitarray('011101'), '5': bitarray('011110'), 'x': bitarr
ay('011111'), '4': bitarray('100000'), '7': bitarray('100001'), '6': bitarray('100010'), 'z': bitar
ray('100011'), 'q': bitarray('100100')}
poprawnie
 albert of prussia 17 may 1490 20 march 1568 was the last grand master of the teutonic knights who
after converting to lutheranism became the first monarch of the duchy of prussia the secularized st
ate that emerged from the former monastic state of t
```

Najkrótsza możliwa długoś kodu dla korpusu to 6 bitów, należy zakodować 37 znaków $2^5 < 37 < 2^6$.

Stopień kompresji wyniesie 1.(3). 8 bitów / 6 bitów

Rozmiar pliku przed kompresją wynosi 2416 kb. Po kompresji pliki mają kod 1 kb a zakodowany tekst 1812 kb. Co daje współczynnik kompresji równy 75.04%.

2. Efektywność kodowania

```
import math
def zadanie2(alfabet):
    entropia = 0
    counter = 0
    for i in alfabet:
        counter+=i[1]
    for i in alfabet:
        entropia += (i[1]/counter * math.log(1/(i[1]/counter),2))
    srednia_długość = 6
    efektywnosc = entropia/srednia_długość
    return entropia,efektywnosc

zadanie2(alfabet)
```

```
(4.2803962467015655, 0.7133993744502609)
```

Dla kodowania z zadania nr 1 entropia wyniosła 4.28, średnia długość słów kodowych była stała i wynosiła 6 a efektywność wyniosła 71.34%.

3. Kodowanie Huffmana

Kodowanie huffmana zostało wykonane dzięki wykorzystaniu funkcji z zadania nr 1. Dodatkowo należało napisać funkcję tworzenia kodowania huffmana oraz funkcje odpowiedzialne za wykonanie kolejnych kroków.

a. Funkcja tworząca kodowanie Huffmana

```
def huffman_decode(node, array = bitarray(0)):
    if type(node) is str:
        return {node: array}
    l = node[0]
    r = node[1]
    d = dict()
    d.update(huffman_decode(l, array + '0'))
    d.update(huffman_decode(r, array + '1'))
    return d

def huffman_code(sign_list):
    nodes = []
    for i in range(len(sign_list)):
        g = sign_list[i]
        nodes.append(g)

    while len(nodes) >1:
        (key1, value1) = nodes[-1]
        (key2, value2) = nodes[-2]
        nodes = nodes[:-2]
        node = (key1, key2)
        nodes.append((node, value1 + value2))
        nodes = sorted(nodes, key=lambda x: x[1], reverse=True)
    huffmanCode = huffman_decode(nodes[0][0])
    return huffmanCode
```

b. Funkcja wyliczająca średnią długość słowa w kodzie

```
def mean_huffman(code,p_list):
    mean_sum = 0
    for i in range(len(code)):
        mean_sum += len(code[i][1])*float(p_list[i][1])
    return(mean_sum)
```

c. Funkcja odpowiedzialna za wykonanie zadania nr 3

```
def zadanie3(code):
    filename = 'norm_wiki_sample.txt'
    filename_save = 'norm_wiki_sample_save.txt'
    filename_code = 'norm_wiki_sample_code.txt'
    znaki = list(string.ascii_lowercase)
    liczby = list(string.digits)
    all_signs = znaki +liczby+ list(" ")

    with open(filename,"r") as f1:
        data = f1.read()

    sign_tracker = {}
    for i in range(len(all_signs)):
        sign_tracker[all_signs[i]] = 0
    counter = 0
    for i in data:
        try:
            sign_tracker[i] += 1
            counter+=1
        except:
            #other signs
            pass

    sign_tracker = dict(sorted(sign_tracker.items(), key=lambda item: item[1], reverse = True))
    sign_list = [(k,v/counter) for k,v in sign_tracker.items()]

    code = huffman_code(sign_list)
    code_list = [(k,v) for k,v in code.items()]
    encoded_text = encode(code,data[0:250])
    save(encoded_text,filename_save,filename_code,code)
    loaded_code,loaded_text = load(filename_code,filename_save)
    decoded_text = decode(loaded_code,loaded_text)
    if data[0:250] == decoded_text:
        msg = "poprawnie"
    else:
        msg = "niepoprawnie"

    p = mean_huffman(code_list,sign_list)
    entropia = zadanie2(alfabet,srednia_długość = p)
    return(code,msg,decoded_text,entropia)

code,msg,decoded_text,entropia = zadanie3('norm_wiki_sample.txt')
print(code)
print(msg)
print(decoded_test)
print(entropia)
```

Wynik zadania trzeciego:

Wynikiem zadania pierwszego jest wygenerowany kod Huffmana, string “poprawnie” bądź “niepoprawnie” w zależności czy odkodowany tekst jest identyczny z tekstem początkowym oraz odkodowany tekst w tym wypadku identyczny tekst jak w pierwszych 250 znakach z pliku 'norm_wiki_sample.txt'.

```
{'e': bitarray('000'), 'm': bitarray('00100'), 'y': bitarray('001010'), 'k': bitarray('0010110'),
'4': bitarray('001011100'), 'x': bitarray('001011101'), '5': bitarray('001011110'), '3': bitarray('00
1011111'), 's': bitarray('0011'), 'w': bitarray('010000'), 'b': bitarray('010001'), 'c': bitarray('01
001'), 'r': bitarray('0101'), 'o': bitarray('0110'), 'n': bitarray('0111'), 'i': bitarray('1000'),
'd': bitarray('10010'), '2': bitarray('10011000'), '9': bitarray('10011001'), 'v': bitarray('100110
1'), 'g': bitarray('100111'), 't': bitarray('1010'), 'p': bitarray('101100'), 'f': bitarray('10110
1'), 'l': bitarray('10111'), 'a': bitarray('1100'), 'h': bitarray('11010'), '8': bitarray('11011000
0'), 'j': bitarray('110110001'), '0': bitarray('11011001'), 'q': bitarray('1101101000'), 'z': bitarra
y('1101101001'), '6': bitarray('1101101010'), '7': bitarray('1101101011'), '1': bitarray('11011011'),
'u': bitarray('110111'), ' ': bitarray('111')}
poprawnie
 albert of prussia 17 may 1490 20 march 1568 was the last grand master of the teutonic knights who af
ter converting to lutheranism became the first monarch of the duchy of prussia the secularized state
that emerged from the former monastic state of t
```

Dodatkowo funkcja ta zwraca entropie i efektywność Huffmana które w tym przypadku wyniosły:

Entropia: 4.28

Efektywność 72.41%

Średnia długość słowa kodu: 5.91

W przypadku kodowania Huffmana średnia długość słowa jest mniejsza niż w przypadku stałej długości, co oznacza, że skompresowany plik będzie mniejszych rozmiarów.