Ekstrakcja informacji

March 20, 2022

[302]:
```
!pip install dahuffman
```

```
Requirement already satisfied: dahuffman in ./venv/lib/python3.8/site-packages
(0.4.1)
```

[303]:
```
import random
from collections import Counter
from dahuffman import HuffmanCodec
```

[304]:
```
NR_INDEKSU = 434784
```

0.1 Wprowadzenie

[305]:
```
tekst = 'Ala ma kota. Jarek ma psa'
```

[306]:
```
codec = HuffmanCodec.from_data(tekst)
```

[307]:
```
Counter(tekst)
```

[307]:
```
Counter({'A': 1,
         'l': 1,
         'a': 6,
         ' ': 5,
         'm': 2,
         'k': 2,
         'o': 1,
         't': 1,
         '.': 1,
         'J': 1,
         'r': 1,
         'e': 1,
         'p': 1,
         's': 1})
```

[308]:
```
codec.print_code_table()
```

```
Bits Code  Value Symbol
   2 00        0 ' '
```

```
  2 01         1 'a'
  4 1000       8 't'
  5 10010     18 _EOF
  5 10011     19 '.'
  5 10100     20 'A'
  5 10101     21 'J'
  5 10110     22 'e'
  5 10111     23 'l'
  4 1100      12 'k'
  4 1101      13 'm'
  5 11100     28 'o'
  5 11101     29 'p'
  5 11110     30 'r'
  5 11111     31 's'
```

[309]:
```
codec.get_code_table()
```

[309]:
```
{' ': (2, 0),
 'a': (2, 1),
 't': (4, 8),
 _EOF: (5, 18),
 '.': (5, 19),
 'A': (5, 20),
 'J': (5, 21),
 'e': (5, 22),
 'l': (5, 23),
 'k': (4, 12),
 'm': (4, 13),
 'o': (5, 28),
 'p': (5, 29),
 'r': (5, 30),
 's': (5, 31)}
```

[310]:
```
encoded = codec.encode(tekst)
```

[311]:
```
"{:08b}".format(int(encoded.hex(),16))
```

[311]:
```
'10100101110100110101001100111001000011001100101010111110101101100001101010011101
11111011'
```

A l a

101001 10111 01

[312]:
```
codec.decode(encoded)
```

[312]:
```
'Ala ma kota. Jarek ma psa'
```

[313]:
```
len(tekst)
```

[313]: 25

[314]:
```
len(encoded)
```

[314]: 11

0.2 Zadanie 1 (15 punktów)

Weź teksty: - z poprzednich zajęć (lub dowolny inny) w języku naturalnym i obetnij do długości 100_000 znaków - wygenerowany losowo zgodnie z rozkładem jednostajnym dyskretnym z klasy [a-zA-Z0-9] o długości 100_000 znaków - wygenerowany losowo zgodnie z rozkładem geometrycznym (wybierz p między 0.2 a 0.8) z klasy [a-zA-Z0-9] o długości 100_000 znaków - wygenerowany losowo zgodnie z rozkładem jednostajnym dwupunktowym p=0.5 z klasy [01] o długości 100_000 znaków - wygenerowany losowo zgodnie z rozkładem jednostajnym dwupunktowym p=0.9 z klasy [01] o długości 100_000 znaków

Następnie dla każdego z tekstów trakując je po znakach: - skompresuj plik za pomocą dowolnego progrmu (zip, tar lub inny) - policz entropię - wytrenuj kodek huffmana i zakoduj cały tekst - zdekoduj pierwsze 3 znaki (jako zera i jedynki) wypisz je (z oddzieleniem na znaki) - zakodowany tekst zapisz do pliku binarnego, zapisz również tablicę kodową - porównaj wielkość pliku tekstowego, skompresowanego pliku tekstowego (zip, ...) oraz pliku skompresowanego hofmmanem (wraz z kodekiem)

Uzupełnij poniższe tabelki oraz wnioski (conajmniej 5 zdań).

0.2.1 START ZADANIA

0.3 Moj tekst mial 90 000 znaków więc obciąłem do tylu wszystkie, szukanie kolejnego + wykonywanie tych wszystkich zadaniań dla 15 punktów nie kalkuluje się względem tego ile czasu trzeba na to wszystko poświęcić.

[315]:
```
!mkdir -p outputs

import requests
from bs4 import BeautifulSoup
txts = [
    ['en', 'https://www.odaha.com/antoine-de-saint-exupery/maly-princ/
 ↪the-little-prince']
]

for k, v in txts:
    soup = BeautifulSoup(requests.get(v).content, "html.parser")
    t = soup.find_all("div", {"class": "entrytext"})
    txt = t[0].get_text()
    text = txt[:90000]
    text = text.replace('\n', ' ' )
```

```
print(f'Fragment: {text[100:1000]}')
print(f'Długość: {len(text)}')
```

```
Fragment: re, about the primeval forest. It was a picture of a boa constrictor
in the act of swallowing an animal. Here is a copy of the drawing.  In the book
it said: "Boa constrictors swallow their prey whole, without chewing it. After
that they are not able to move, and they sleep through the six months that they
need for digestion." I pondered deeply, then, over the adventures of the jungle.
And after some work with a colored pencil I succeeded in making my first
drawing. My Drawing Number One. It looked something like this:  I showed my
masterpiece to the grown-ups, and asked them whether the drawing frightened
them. But they answered: "Frighten? Why should any one be frightened by a hat?"
My drawing was not a picture of a hat. It was a picture of a boa constrictor
digesting an elephant. But since the grown-ups were not able to understand it, I
made another drawing: I drew the inside of a boa
Długość: 90000
```

[316]:
```
import string
import numpy as np
size = 90000
alphabet = string.ascii_lowercase + string.ascii_uppercase + string.digits
print(alphabet)
generated = {}
generated['ksiazka'] = text
```

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

- wygenerowany losowo zgodnie z rozkładem jednostajnym dyskretnym z klasy [a-zA-Z0-9] o długości 90_000 znaków

[317]:
```
uniform = ''.join(np.random.choice(list(alphabet), size))
uniform[100:200]
generated['jednostajny'] = uniform
```

- wygenerowany losowo zgodnie z rozkładem geometrycznym (wybierz p między 0.2 a 0.8) z klasy [a-zA-Z0-9] o długości 90_000 znaków

[318]:
```
g = np.random.geometric(p=0.6, size=size) - 1
geometric = ''.join([(alphabet)[g[i]] for i in range(size)])
geometric[100:200]
generated['geometryczny'] = geometric
```

- wygenerowany losowo zgodnie z rozkładem jednostajnym dwupunktowym p=0.5 z klasy [01] o długości 90_000 znaków

[319]:
```
c = [0, 1]
uniform_5 = ''.join(map(str, np.random.choice(c, size, 0.5)))
```

```
uniform_5[100:200]
generated['jednostajny_5'] = uniform_5
```

- wygenerowany losowo zgodnie z rozkładem jednostajnym dwupunktowym p=0.9 z klasy [01] o długości 90_000 znaków

[320]:
```
uniform_9 =  ''.join(map(str, np.random.binomial(1, 0.9, size)))
uniform_9[100:200]
generated['jednostajny_9'] = uniform_9
```

0.4 Kompresja

[321]:
```
!mkdir -p outputs
```

[322]:
```
from zipfile import ZipFile
import zlib
from collections import Counter
import math

def uncompress(name, text):
    print(f'{name} wielkość {len(text)}')
    ent = entropy(text)
    print(f'{name} entropia {ent}')

def save_text(name, content):
    with open(f"outputs//{name}.txt", "w",  encoding="utf-8") as f:
        f.write(content)

def compress(name, text):
    # zipObj = ZipFile(f'outputs//{name}.zip', 'w')
    # zipObj.write(f'outputs//{name}.txt')
    # print(sum([zinfo.file_size for zinfo in zipObj.filelist]))
    compressed = zlib.compress(text)
    print(f'{name} wielkość po skompresowaniu {len(compressed)}')
    ent = 8 * len(compressed) / len(text)
    print(f'{name} entropia po skompresowaniu {ent}')
    with open(f'outputs//{name}.zip', "wb") as f:
        f.write(compressed)

def get_characters(t):
    yield from t

def entropy(text):
    result=0
    x=Counter(get_characters(text))
    for i in x:
        x[i]=x[i]/len(text)
```

```
        y=x[i]*math.log2(x[i])
        result+=y
    result=result*(-1)
    return result
```

[323]:
```
for key in generated:
    save_text(key, generated[key])
    uncompress(key, generated[key])
    compress(key, bytes(generated[key], 'utf-8'))
    print("")
```

```
ksiazka wielkość 90000
ksiazka entropia 4.387839789390687
ksiazka wielkość po skompresowaniu 31504
ksiazka entropia po skompresowaniu 2.8000755480007555

jednostajny wielkość 90000
jednostajny entropia 5.953598358092162
jednostajny wielkość po skompresowaniu 67703
jednostajny entropia po skompresowaniu 6.0180444444444445

geometryczny wielkość 90000
geometryczny entropia 1.6198960394407873
geometryczny wielkość po skompresowaniu 24404
geometryczny entropia po skompresowaniu 2.1692444444444443

jednostajny_5 wielkość 90000
jednostajny_5 entropia 0.9999998845843936
jednostajny_5 wielkość po skompresowaniu 14383
jednostajny_5 entropia po skompresowaniu 1.2784888888888888

jednostajny_9 wielkość 90000
jednostajny_9 entropia 0.4714913357310381
jednostajny_9 wielkość po skompresowaniu 8603
jednostajny_9 entropia po skompresowaniu 0.7647111111111111
```

- wytrenuj kodek huffmana i zakoduj cały tekst
- zdekoduj pierwsze 3 znaki (jako zera i jedynki) wypisz je (z oddzieleniem na znaki)
- zakodowany tekst zapisz do pliku binarnego, zapisz również tablicę kodową

[324]:
```
from dahuffman import HuffmanCodec
from collections import Counter


def huffman(text, name):
    c = HuffmanCodec.from_data(text)
```

```
    c.save(f'outputs//{name}-codec.bin')
    c.print_code_table()
    c.get_code_table()
    encoded = c.encode(text)
    huff_code = "{:08b}".format(int(encoded.hex(), 16))

    print(huff_code[:10])
    decoded = codec.decode(encoded)
    print(f'{decoded[0]} {decoded[1]} {decoded[2]}')

    f = open(f'outputs//{name}.bin', "wb")
    f.write(bytes(encoded))
    f.close()
```

[325]:
```
for key in generated:
    print(key)
    huffman(generated[key], key)
```

```
ksiazka
Bits Code               Value Symbol
   2 00                     0 ' '
   3 010                    2 'e'
   6 011000                24 'p'
   7 0110010               50 'k'
   8 01100110             102 '-'
   9 011001110            206 '?'
  10 0110011110           414 "'"
  11 01100111110          830 ';'
  11 01100111111          831 'F'
   6 011010                26 'g'
   6 011011                27 '"'
   4 0111                   7 'o'
   4 1000                   8 'a'
   6 100100                36 'f'
   6 100101                37 'c'
   5 10011                 19 'd'
   7 1010000               80 'v'
   7 1010001               81 'I'
   6 101001                41 'y'
   6 101010                42 'w'
   6 101011                43 '.'
   4 1011                  11 't'
   5 11000                 24 'l'
   7 1100100              100 'b'
  10 1100101000           808 'Y'
  14 11001010010000     12944 '9'
  16 1100101001000100   51780 'ě'
```

```
16 1100101001000101   51781 'ů'
15 110010100100011    25891 '4'
14 11001010010010     12946 '('
14 11001010010011     12947 ')'
12 110010100101        3237 'D'
15 110010100110000    25904 '7'
15 110010100110001    25905 'U'
14 11001010011001     12953 'J'
13 1100101001101       6477 '1'
13 1100101001110       6478 '2'
13 1100101001111       6479 'L'
10 1100101010           810 ':'
10 1100101011           811 'H'
10 1100101100           812 'W'
13 1100101101000       6504 'P'
16 1100101101001000   52040 '…'
17 11001011010010010 104082 _EOF
17 11001011010010011 104083 '$'
15 110010110100101    26021 '5'
17 11001011010011000 104088 '8'
17 11001011010011001 104089 'K'
16 1100101101001101   52045 'R'
16 1100101101001110   52046 'V'
17 11001011010011110 104094 'Z'
17 11001011010011111 104095 '¨'
12 110010110101        3253 'E'
11 11001011011         1627 'S'
 9 110010111            407 'A'
 6 110011                51 'm'
 5 11010                 26 'r'
 6 110110                54 'u'
 7 1101110              110 ','
10 1101111000           888 '!'
10 1101111001           889 'B'
12 110111101000        3560 'z'
13 1101111010010       7122 'C'
14 11011110100110     14246 '3'
15 110111101001110    28494 'ď'
15 110111101001111    28495 '6'
11 11011110101         1781 'q'
11 11011110110         1782 'j'
11 11011110111         1783 'O'
 9 110111110            446 'T'
12 110111111000        3576 'O'
12 110111111001        3577 'G'
12 110111111010        3578 'M'
12 110111111011        3579 'N'
10 1101111111           895 'x'
```

```
   5 11100                28 's'
   5 11101                29 'i'
   5 11110                30 'h'
   5 11111                31 'n'
1101111011
    m
jednostajny
Bits Code    Value Symbol
   5 00000       0 'Z'
   5 00001       1 't'
   7 0001000     8 _EOF
   7 0001001     9 '1'
   6 000101      5 'W'
   6 000110      6 'C'
   6 000111      7 'r'
   6 001000      8 'j'
   6 001001      9 'w'
   6 001010     10 'M'
   6 001011     11 'm'
   6 001100     12 'H'
   6 001101     13 'R'
   6 001110     14 'v'
   6 001111     15 'L'
   6 010000     16 'P'
   6 010001     17 'I'
   6 010010     18 'f'
   6 010011     19 'B'
   6 010100     20 'd'
   6 010101     21 'G'
   6 010110     22 'e'
   6 010111     23 '0'
   6 011000     24 'V'
   6 011001     25 '8'
   6 011010     26 'a'
   6 011011     27 'q'
   6 011100     28 '6'
   6 011101     29 '9'
   6 011110     30 'A'
   6 011111     31 's'
   6 100000     32 '3'
   6 100001     33 'i'
   6 100010     34 '7'
   6 100011     35 'n'
   6 100100     36 'E'
   6 100101     37 'K'
   6 100110     38 'o'
   6 100111     39 'X'
   6 101000     40 'F'
```

```
   6 101001     41 'S'
   6 101010     42 'k'
   6 101011     43 'l'
   6 101100     44 'h'
   6 101101     45 'Y'
   6 101110     46 'z'
   6 101111     47 'x'
   6 110000     48 'u'
   6 110001     49 'g'
   6 110010     50 'U'
   6 110011     51 'p'
   6 110100     52 'J'
   6 110101     53 '5'
   6 110110     54 'Q'
   6 110111     55 '4'
   6 111000     56 'c'
   6 111001     57 'T'
   6 111010     58 'b'
   6 111011     59 'D'
   6 111100     60 '2'
   6 111101     61 'O'
   6 111110     62 'y'
   6 111111     63 'N'
1010110100
    J
geometryczny
Bits Code           Value Symbol
  13 0000000000000     0 _EOF
  13 0000000000001     1 'p'
  12 000000000001      1 'm'
  11 00000000001       1 'k'
  10 0000000001        1 'j'
   9 000000001         1 'i'
   8 00000001          1 'h'
   7 0000001           1 'g'
   6 000001            1 'f'
   5 00001             1 'e'
   4 0001              1 'd'
   3 001               1 'c'
   2 01                1 'b'
   1 1                 1 'a'
1111110011
a s
jednostajny_5
Bits Code Value Symbol
   2 00       0 _EOF
   2 01       1 '1'
   1 1        1 '0'
```

```
1110101101
p a e
jednostajny_9
Bits Code Value Symbol
   2 00       0 _EOF
   2 01       1 '0'
   1 1        1 '1'
1111111101
s p s
```

- porównaj wielkość pliku tekstowego, skompresowanego pliku tekstowego (zip, ...) oraz pliku skompresowanego hofmmanem (wraz z kodekiem)

[326]:
```
import os
def compare_all(name):
    s_txt = os.path.getsize(f'outputs//{name}.txt')
    print(f'PLAIN {name}.txt wielkość {s_txt}')
    s_zip = os.path.getsize(f'outputs//{name}.zip')
    print(f'COMPRESSED {name}.zip wielkość {s_zip}')
    s_hof = os.path.getsize(f'outputs//{name}.bin') + os.path.getsize(f'outputs/
 ↪/{name}-codec.bin')
    print(f'HOFFMAN {name}.bin + {name}.codec wielkość {s_hof}')
    print('')

for key in generated:
    print(key)
    compare_all(key)
```

```
ksiazka
PLAIN ksiazka.txt wielkość 90009
COMPRESSED ksiazka.zip wielkość 31504
HOFFMAN ksiazka.bin + ksiazka.codec wielkość 50884

jednostajny
PLAIN jednostajny.txt wielkość 90000
COMPRESSED jednostajny.zip wielkość 67703
HOFFMAN jednostajny.bin + jednostajny.codec wielkość 68067

geometryczny
PLAIN geometryczny.txt wielkość 90000
COMPRESSED geometryczny.zip wielkość 24404
HOFFMAN geometryczny.bin + geometryczny.codec wielkość 19056

jednostajny_5
PLAIN jednostajny_5.txt wielkość 90000
COMPRESSED jednostajny_5.zip wielkość 14383
HOFFMAN jednostajny_5.bin + jednostajny_5.codec wielkość 17053
```

```
jednostajny_9
PLAIN jednostajny_9.txt wielkość 90000
COMPRESSED jednostajny_9.zip wielkość 8603
HOFFMAN jednostajny_9.bin + jednostajny_9.codec wielkość 12564
```

Entropia

	Entropia
tekst w jęz. naturalnym	4.387839789390687
losowy tekst (jednostajny)	5.953701289211957
losowy tekst (geometryczny)	1.6246104667414993
losowy tekst (dwupunktowy 0.5)	0.9999683659193007
losowy tekst (dwupunktowy 0.9)	0.4714562533788368

Wielkości w bitach:

	Plik nieskompresowany	Plik skompresowany (zip, tar,..)	Plik zakodowany + tablica kodowa
tekst w jęz. naturalnym	90009	31504	50884
losowy tekst (jednostajny)	90009	67694	68072
losowy tekst (geometryczny)	90009	24465	19095
losowy tekst (dwupunktowy 0.5)	90009	14386	17018
losowy tekst (dwupunktowy 0.9)	90009	8584	12564

Wnioski: Moim głównym wnioskiem jest to, że nie rozumiem po co mamy tutaj się rozpisywać, skoro uzupełnialiśmy tabelki idealnie ukazujące wyniki zadań. Myśle że dla przyszłych roczników dużo ciekawszym by było omówienie albo napisanie np dlaczego (na windowsie) jak wejdziemy sobie w właściwości pliku txt, dostajemy dwie informacje “Size” i “Size on disk”. Ale skoro trzeba tutaj coś napisać i polać troche wody to: - Wszystkie teksty nieskopresowane zapisane do pliku txt posiadają taką samą wielkość. - Tekst naturalny po skompresowaniu zajął 1/3 miejsca - Tekst wygenerowany jednostajnie po skompresowaniu zajął około 2/3 miejsca - Tekst wygenerowany geometrycznie po skompresowaniu zajął około 1/4 miejsca - Tekst wygenerowany dwupunktowy 0.5 po skompresowaniu zajął około 2/6 miejsca - Tekst wygenerowany dwupunktowy 0.9 po skompresowaniu zajął około 1/10 miejsca - Tekst naturalny po zakodowaniu hoffmanem zajął wiecej miejsca niz po skompresowaniu - Tekst wygenerowany jednostajnie po zakodowaniu hoffmanem zajął mniej wiecej tyle samo miejsca co po skompresowaniu - Tekst wygenerowany geometrycznie po zakodowaniu hoffmanem zajął mniej miejsca niz po skompresowaniu - Tekst wygenerowany dwupunktowy 0.5 po zakodowaniu hoffmanem zajął wiecej miejsca niz po skompresowaniu - Tekst wygenerowany dwupunktowy 0.9 po zakodowaniu hoffmanem zajął wiecej miejsca niz po skompresowaniu

Dla moich wyników cała zabawa z Hoffmanem była w większości przypadków mało efektowna, ponieważ plik bin z kodekiem zajął w 4/5 przypadków więcej miejsca niż pierwsza funkcja pythonowa która wyskoczyła mi w googlu służąca do kompresji.

0.4.1 KONIEC ZADANIA

0.5 Zadanie 2 (10 punktów)

Powtórz kroki z zadania 1, tylko potraktuje wiadomości jako słowa (oddzielone spacją). Jeżeli występują więcej niż jedna spacja równocześnie- usuń je.

Do wniosków dopisz koniecznie porównanie między kodowaniem hoffmana znaków i słów.

0.5.1 START ZADANIA

[327]:
```
a = list(" " + string.ascii_lowercase + string.ascii_uppercase + string.digits)
a
```

[327]:
```
[' ',
 'a',
 'b',
 'c',
 'd',
 'e',
 'f',
 'g',
 'h',
 'i',
 'j',
 'k',
 'l',
 'm',
 'n',
 'o',
 'p',
 'q',
 'r',
 's',
 't',
 'u',
 'v',
 'w',
 'x',
 'y',
 'z',
 'A',
 'B',
 'C',
 'D',
```

```
 'E',
 'F',
 'G',
 'H',
 'I',
 'J',
 'K',
 'L',
 'M',
 'N',
 'O',
 'P',
 'Q',
 'R',
 'S',
 'T',
 'U',
 'V',
 'W',
 'X',
 'Y',
 'Z',
 '0',
 '1',
 '2',
 '3',
 '4',
 '5',
 '6',
 '7',
 '8',
 '9']
```

[328]:
```
discret = ''.join(np.random.choice(list(a), 90000))
print(discret[100:200])

g = np.random.geometric(p=0.2, size=90000)-1
geometric = ''.join([(a)[g[i]] for i in range(90000)])
print(geometric[100:200])
```

```
31Swf12sLNxttpTMTsNOiClvfv1cWI3mykXEQv7XDtmpeZbE8Lqs
gRZU5VaLnYixPZTn4gTsZxVO8xjH8BQCx1kA2m6AveQHonr
baeesh  acyjb cb  a bbak a b  xjfhcqw b af aib kbh ec abb afedehfedheafia aabd
aacedcefbb    e  b d
```

[329]:
```
def remove_extra_space(text):
    return " ".join(text.split())
```

```
def entropy(text):
    result=0
    x=Counter(nltk.tokenize.word_tokenize(text))
    for i in x:
        x[i]=x[i]/len(text)
        y=x[i]*math.log2(x[i])
        result+=y
    result=result*(-1)
    return result

generated2 = {}
generated2['ksiazka2'] = remove_extra_space(text)
generated2['dyskretny2'] = remove_extra_space(discret)
generated2['geometryczny2'] = remove_extra_space(geometric)
```

[330]:
```
for key in generated2:
    save_text(key, generated2[key])
    uncompress(key, generated2[key])
    compress(key, bytes(generated2[key], 'utf-8'))
```

```
ksiazka2 wielkość 89883
ksiazka2 entropia 2.421775520775303
ksiazka2 wielkość po skompresowaniu 31391
ksiazka2 entropia po skompresowaniu 2.793663507319895
dyskretny2 wielkość 89974
dyskretny2 entropia 0.2614291392553716
dyskretny2 wielkość po skompresowaniu 67867
dyskretny2 entropia po skompresowaniu 6.03436548336186
geometryczny2 wielkość 86421
geometryczny2 entropia 2.325385770070672
geometryczny2 wielkość po skompresowaniu 45978
geometryczny2 entropia po skompresowaniu 4.256187732148436
```

[331]:
```
import nltk
def huffman(tekst, name):
    codec = HuffmanCodec.from_data(nltk.tokenize.word_tokenize(tekst))
    codec.save(f'outputs//{name}-codec.bin')
    encoded = codec.encode(nltk.tokenize.word_tokenize(tekst))
    with open(f'outputs//{name}.bin',"wb") as f:
        f.write(bytes(encoded))

def first3(text):
    codec = HuffmanCodec.from_data(nltk.tokenize.word_tokenize(text))
    print(f' {nltk.tokenize.word_tokenize(text)[0]} -> {int(codec.encode([nltk.
 ↪tokenize.word_tokenize(text)[0]]).hex(), 16)}')
```

```
    print(f' {nltk.tokenize.word_tokenize(text)[1]} -> {int(codec.encode([nltk.
↪tokenize.word_tokenize(text)[1]]).hex(), 16)}')
    print(f' {nltk.tokenize.word_tokenize(text)[2]} -> {int(codec.encode([nltk.
↪tokenize.word_tokenize(text)[2]]).hex(), 16)}')
```

[332]:
```
for key in generated2:
    print(key)
    huffman(generated2[key], key)
    first3(generated2[key])
```

```
ksiazka2
 Once -> 1023
 when -> 21372
 I -> 71
dyskretny2
 mE1Tpr9lDdbYO8yNSy23S6J28mMGGuzcPQoWzpey4PS -> 18982
 goDkf -> 10406
 q1vjStFXVKCIkTjs3ymejZAyNSMxBNvZunh -> 25254
geometryczny2
 ghbai -> 54485
 mgejai -> 60633
 ifa -> 17478
```

[333]:
```
for key in generated2:
    print(key)
    compare_all(key)
```

```
ksiazka2
PLAIN ksiazka2.txt wielkość 89892
COMPRESSED ksiazka2.zip wielkość 31391
HOFFMAN ksiazka2.bin + ksiazka2.codec wielkość 59032

dyskretny2
PLAIN dyskretny2.txt wielkość 89974
COMPRESSED dyskretny2.zip wielkość 67867
HOFFMAN dyskretny2.bin + dyskretny2.codec wielkość 104634

geometryczny2
PLAIN geometryczny2.txt wielkość 86421
COMPRESSED geometryczny2.zip wielkość 45978
HOFFMAN geometryczny2.bin + geometryczny2.codec wielkość 167430
```

Entropia

	Entropia
tekst w jęz. naturalnym	2.421775520775303
losowy tekst (dyskretny)	0.25185877675081064

	Entropia
losowy tekst (geometryczny)	2.3333186184230708

Wielkości w bitach:

	Plik nieskompresowany	Plik skompresowany (zip, tar,..)	Plik zakodowany + tablica kodowa
tekst w jęz. naturalnym	89892	31391	59032
losowy tekst (jednostajny)	89974	67867	104634
losowy tekst (geometryczny)	86421	45978	167430

Wnioski:

- Pliki teskstowe różnią się wielkością bo usuwaliśmy powielone spacje.
- Tekst naturalny po skompresowaniu zajął 1/3 miejsca
- Tekst wygenerowany jednostajnie po skompresowaniu zajął około 2/3 miejsca
- Tekst wygenerowany geometrycznie po skompresowaniu zajął około 1/2 miejsca
- Tekst naturalny po zakodowaniu hoffmanem zajął wiecej miejsca niz po skompresowaniu
- Tekst wygenerowany jednostajnie po zakodowaniu hoffmanem zajął mniej wiecej tyle samo miejsca co po skompresowaniu
- Tekst wygenerowany geometrycznie po zakodowaniu hoffmanem zajął mniej miejsca niz po skompresowaniu
- Zakodowany tekst hoffmanem jednostajny zajął więcej miejsca niż zwykły plik tekstowy (!)
- Zakodowany ekst hoffmanem geometryczny zajął DWUKROTNIE więcej miejsca niż zwykły plik tekstowy (!)

0.5.2 KONIEC ZADANIA

0.6 Zadanie 3 (20 punktów)

stwórz ręcznie drzewo Huffmana (zrób rysunki na kartce i załącz je jako obrazek) oraz zakoduj poniższy tekst

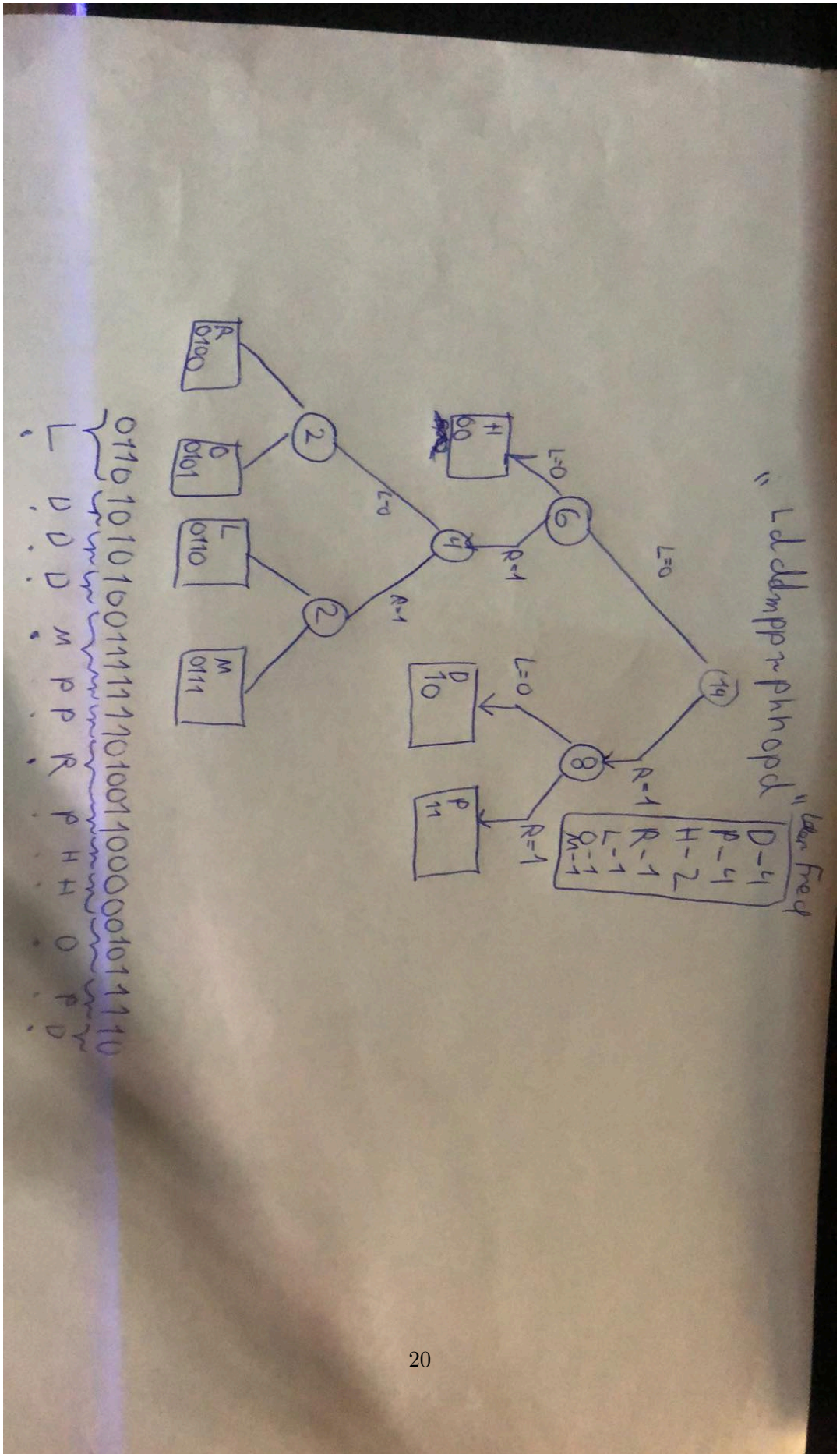
[334]:
```
random.seed(123)

tekst = list('abcdefghijklmnoprst')

random.shuffle(tekst)

tekst = tekst[: 5 + random.randint(1,5)]
```

```
tekst = [a*random.randint(1,4) for a in tekst]

tekst = [item for sublist in tekst for item in sublist]

''.join(tekst)

random.shuffle(tekst)

tekst = ''.join(tekst)
```

[335]:
```
tekst
```

[335]: 'ldddmpprphhopd'

0.6.1 Start zadania



0.6.2 Koniec zadania

0.7 WYKONANIE ZADAŃ

Zgodnie z instrukcją 01_Kodowanie_tekstu.ipynb