Telekomunikacja zadanie 3 – Kodowanie Huffmana

Paweł Bucki 224270
Krzysztof Woźniakowski 224460

Funkcje Klienta:

```
import socket
from Node import Node
#ustawienenia naszego gniazda, port oraz nagłówek domyslny
HEADER = 64
PORT = 5050
FORMAT = 'utf-8' #formatowanie tekstu
DISCONNECT_MESSAGE = "!DISCONNECT" #formatowanie tekstu
SERVER = "10.9.25.109" # ip klienta
ADDR = (SERVER, PORT)

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  #tworzenie gniazda
client.connect(ADDR)  #przypisanie adresu

def send(msg): #funkcja osblugujaca wysylanie informacji gniazdem
    message = msg.encode(FORMAT) #deklaracja formatowania tekstu
    msg_length = len(message) #okreslenei dlugosci tekstu
    send_length = str(msg_length).encode(FORMAT) #nadanie dlugosci
    send_length += b' ' * (HEADER - len(send_length)) #dodanie naglowka
    client.send(send_length) #funkcja wysylajaca pakiet z dlugoscia wiadomosci
    client.send(message) #funkcja wysylajaca wiadomosc
    print(client.recv(2048).decode(FORMAT)) #potiwerdznei wyslania

nodes = []
initialNodes = []
def codeMessage(message): #kodowanie drzewa
    #sortowanie wg czestototliwosci  oraz zlicznie znkaow
    dictionary = {}
    for char in message:
        if not char in dictionary :
            dictionary[char] = 1
        else :
            dictionary[char] = dictionary[char] + 1

    sortedDictionary = sorted(dictionary.items(),key=lambda x:x[1])
    print(sortedDictionary)

    for char in sortedDictionary :
        initialNodes.append( Node( sortedDictionary[char] ) ) #tworzenie drzewa

    nodes.append(initialNodes[0])
    for i in range ( 1, len(initialNodes) ) :
        if initialNodes[i].value > nodes[i-1].value :
            nodes.append( Node( nodes[-1].value + initialNodes[i].value, nodes[-2], nodes[-1] ) )


string = 'Algorytm Huffmana działa rewelacyjnie!'

class NodeTree(object):
    def __init__(self, left=None, right=None):
        self.left = left #przejscie na najblizsza lewa galaz
        self.right = right #przejscie na najblizsza prawa galaz
```

```
    def children(self): #definicja "potomka"
        return (self.left, self.right)
    def nodes(self): #definicja wezla
        return (self.left, self.right)
    def __str__(self): #funkcja przechowujaca wartosc dla danego wezla
        return '%s_%s' % (self.left, self.right)

def huffmanCodeTree(node, left=True, binString=''): #implementacja drzewa oraz kodowania huffmana
    if type(node) is str:
        return {node: binString} #zamiana na binarny string odpowiedni do przeslania gniazdem
    (l, r) = node.children() #deklaracja drzewa
    d = dict() #deklaracja slownika
    d.update(huffmanCodeTree(l, True, binString + '0')) # przypisanie 0 przy przejsciu na lewo
    d.update(huffmanCodeTree(r, False, binString + '1'))  # przypisanie 1 przy przejsciu na prawo
    return d

#sprawdzanie czestotliwosci wystepowania danego znaku w wiadomosci do zakodownaia
freq = {}
for c in string:
    if c in freq:
        freq[c] += 1
    else:
        freq[c] = 1

freq = sorted(freq.items(), key=lambda x: x[1], reverse=True) #sortowanie wg czestotliwosci
nodes = freq #przepisanie wartosci

while len(nodes) > 1: #petla dla całego drzewa, tworzenei slownika
    (key1, c1) = nodes[-1] #rozprowadzanie danych po drzewie
    (key2, c2) = nodes[-2]
    nodes = nodes[:-2]
    node = NodeTree(key1, key2)
    nodes.append((node, c1 + c2))
    nodes = sorted(nodes, key=lambda x: x[1], reverse=True)

huffmanCode = huffmanCodeTree(nodes[0][0]) #przekazanie slownika i danych do zmiennych
print(' Znak - Kod Huffmana ') #wyswietlanie

print(huffmanCode) #wysiwetlanie zakodownaj wiadomosci
messageToSend = ''
dictionaryToSend = '/'
for char in string: #dodanie po znaku naszego kodu
    messageToSend = messageToSend + huffmanCode[char]

#Przepisanie slownika na string
for char in huffmanCode :
    dictionaryToSend = dictionaryToSend + char + ':' + huffmanCode[char] + ','

print(dictionaryToSend) #wyswietlanie slownika
print(messageToSend) #wyswietlanie wiadomosci

send(dictionaryToSend) #wysylanie slownika
send(messageToSend) #wysylanie wiadomosci

send(DISCONNECT_MESSAGE) #koniec polaczneia
```

Funkcje serwera z gniazdem

```
import socket
import threading
#ustawienenia naszego gniazda, port oraz nagłówek domyslny
HEADER = 64
PORT = 5050
SERVER = socket.gethostbyname(socket.gethostname())
ADDR = (SERVER, PORT)
FORMAT = 'utf-8' #formatowanie tekstu
DISCONNECT_MESSAGE = "!DISCONNECT"#formatowanie tekstu

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #tworzenie gniazda
server.bind(ADDR) #przypisanie adresu

dictionary = {} #deklaracja slownika do odebrnia

def handle_message_dict( message ) : #funkcja odbierająca i interpretujaca slownik
    i = 1
    dict = {}
    while i < len(message)- 1 : #podzial slownika na bity, odczytanie znaków
        dict[ message[i] ] = ''
        j = i
        i = i+2
        while message[i] is not ',' :
            dict[ message[j] ] = dict[ message[j] ] + message[i] #przepisywanie odczytanych znakow do
slownika
            i = i + 1
        i = i + 1
    print('Otrzymany slownik: ' , dict) #wyswietlenie zawartosci
    return dict

def translate(message,dict) : #funkjca tlumaczaca odebrane dane wg slownika
    solution = ''
    str = ''
    for char in message :
        str = str + char
        for element in dict :
            if dict[element] == str : #porownainie bitowej postaci znaku do znakow ze slownika
                solution = solution + element
                str = ''
    print ( 'Przetlumaczone ' + solution ) #wyswietlanie zawartosci


def handle_client(conn, addr): #funkcja obslugujaca polaczenie gniazda
    print(f"[NEW CONNECTION] {addr} connected.")

    connected = True #deklaracja otwartego polaczenia
    while connected: #nastawienei na odbior podczas pracy programu
        msg_length = conn.recv(HEADER).decode(FORMAT)
        if msg_length:
            msg_length = int(msg_length)
            msg = conn.recv(msg_length).decode(FORMAT)
            if msg == DISCONNECT_MESSAGE: #zamkniecie polaczenia po otrzymaniu wiadomosci
                connected = False

            #print(f"[{addr}] {msg}")
            conn.send("Msg received".encode(FORMAT)) #potwierdzenie odbioru wiadomosci
```

```
            if(msg[0] == '/') : #odbieranie slownika
                dictionary = handle_message_dict(msg)
            else: #odbieranie wiadomosci i jej tlumaczenie
                translate(msg,dictionary)

    conn.close() #zamkniecie polaczenia


def start():
    server.listen() #start nasluchiwania
    print(f"[LISTENING] Server is listening on {SERVER}")
    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handle_client, args=(conn, addr)) #rozpoczecie watku polaczenia
        thread.start()
        print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}") #wyswietlanie aktywnych polaczen


print("[STARTING] server is starting...")
start() #wywolanie funkcji staru porgramu odbierajacego dane
```

Klasa dzrewa

```
class Node :
    def __init__(self,value,left = None,right = None):
        self.value = value
        self.left = left
        self.right = right
```