| Telekomunikacja - laboratorium | | | | Studia dzienne - inżynierskie | |
|---|---|---|---|---|---|
| **Nazwa zadania** | **Algorytm statycznego kodowania Huffmana** | | | | |
| **Dzień** | **czwartek** | **Godzina** | **12:00** | **Rok akademicki** | **2021/2022** |
| **Imię i Nazwisko** | **Szymon Habrych** | | | | |
| **Imię i Nazwisko** | **Bartosz Drągowski** | | | | |
| **Imię i Nazwisko** | | | | | |

**Opis programu, rozwiązania problemu.**

**Program służy do przesyłania wiadomości tekstowych pomiędzy komputerami, za pomocą interfejsu gniazd sieciowych i protokołu TCP.**
**Aplikacja pozwala na wybór trybu pracy za pomocą TUI, gdzie jeden to wysłanie wiadomości, a drugi to jej odbiór, po wcześniejszym podaniu adresu IP server'a(IPv4 komputera na który chcemy wysłać wiadomość), a następnie tworzone jest gniazdo sieciowe. W przypadku odbioru wiadomości ustawione zostaje ono do nasłuchu.**

**Aby wysłać wiadomość trzeba podać IP i port server'a, na który wiadomość powinna zostać wysłana.**

**Kodowanie wiadomości odbywa się przy pomocy algorytmu kodowania Huffmana. Słownik kodowy tworzony jest na podstawie tekstu wiadomości i może zawierać każdy znak z UTF-8.**

**Tworzenie drzewa kodowego:**
**Realizowany przez funkcje make_tree. Zliczone i uporządkowane zostają w kolejności niemalejącej wystąpienia poszczególnych znaków w wiadomości. Na początku z końca listy zostają pobrane dwa elemnety(gdzie zawsze rzadziej występujący element w drzewie będzie po jego lewej stronie) i na ich podstawie tworzony jest pierwszy węzeł drzewa, gdzie dziećmi są pobrane elementy, a rodzicem jest suma częstości wystąpień tych elementów. Każdy kolejny węzeł drzewa będzie zawierać kolejny element z końca listy, rodzica poprzedniego węzła i nowym rodzicem będzie suma wystąpień tych elementów. Algorytm będzie się wykonywać aż do wyczerpania elementów w liście.**

**Kodowanie wiadomości:**
**Realizowane przez funkcję encode_message. Jako argumenty przyjmuje wiadomość i słownik stworzony na bazie drzewa Huffmana. Każdy znak w tekście jest przekształcany na ciąg bitów będących „ścieżką" prowadzącą z korzenia drzewa kodowego, do węzła zawierającego zakodowany znak. 0 w tym wypadku oznacza przejście do lewego węzła, 1 do prawego. Po zakodowaniu, wszystkie bity są kompresowane i zwracane w formie wiadomości.**

**Dekodowanie wiadomości:**
**Realizowane przez funkcję decode_message. Pobierana zostaje wiadomość złożona z 0 i 1, jako ciąg bitów. Następnie sprawdzane są kolejne bity wiadomości. Pobrany zostaje 1. bit wiadomości. Jeżeli nie ma swojego odpowiednika w słowniku, dodawany na jego koniec zostaje 2. bit wiadomości i znowu przeszukiwany zostaje słownik w celu znalezienia odpowiednika takiego ciągu bitów. Takie dodawanie bitów trwa, aż do momentu znalezienia odpowiednika ich ciągu w słowniku. Do wiadomości zwrotnej zostaje dodany odnaleziony znak, a algorytm powtarza czynność, rozpoczynając poszukiwania od jeszcze nie sprawdzonego pierwszego bitu i trwa aż do wyczerpania wszystkich bitów wiadomości. Odkodowana wiadomość zostaje zwrócona przez funkcje, a następnie zapisana do pliku.**

**Najważniejsze elementy kodu programu z opisem.**

```
#klasa pojedynczego węzła drzewa
class NodeTree(object):
    def __init__(self, left=None, right=None):
        self.left = left
        self.right = right

    def children(self):
        return self.left, self.right

    def __str__(self):
        return self.left, self.right


#funkcja znajdująca odpowiednie kodowania w drzewie i zapisująca je do słownika
def huffman_code_tree(node, bin_string=''):
    if type(node) is str:
        return {node: bin_string}
    (l, r) = node.children()
    dictionary = dict()
    dictionary.update(huffman_code_tree(l, bin_string + '0'))
    dictionary.update(huffman_code_tree(r, bin_string + '1'))
    return dictionary
```

```python
#funkcja tworząca drzewo binarne Huffmana
def make_tree(nodes):
    while len(nodes) > 1:
        (key1, c1) = nodes[-1]
        (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)
    return nodes[0][0]

#wyznaczanie słownika opartego na kodowaniu Huffmana
def encoding(message: str):
    freq = dict(Counter(message))
    #sortowanie znaków od najczęściej występującego do najrzadziej występującego
    freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)
    node = make_tree(freq)
    return huffman_code_tree(node)
```

```python
#zwraca słownik stworzony na bazie wiadomości przy użyciu drzewa Huffmana
def create_dictionary(message: str):
    return hc.encoding(message)


#kodowanie wiadomości, zmiana znaku wiadomości na odpowiednik w drzewie
def encode_message(message, dictionary):
    encoded_message = ''
    for word in message:
        encoded_message += dictionary[word]
    return bytes(encoded_message, 'utf-8')

#dekodowanie wiadomości, znajdowanie odpowiednich znaków w słowniku
def decode_message(encoded_message: str, dictionary):
    message = ''
    word = ''
    for i in range(len(encoded_message)):
        word += encoded_message[i]
        try:
            message += find_value_in_dictionary(word, dictionary)
            word = ''
        except Exception:
            pass
    return message


#odnajdowanie znaków na podstawie ciągu 0 i 1
def find_value_in_dictionary(code: str, dictionary):
    for word, dic_code in dictionary.items():
        if dic_code == code:
            return word
    raise Exception
```

```python
#funkcja odpowiedzialna za wysłanie wiadomości
def send(message, dictionary, ip: str, port: int):
    for i in range(10):
        time.sleep(1)
        try:
            # tworzenie socketu klienta
            client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            # łączenie się z serverem
            client_socket.connect((ip, port))
            dictionary_length = len(dictionary)
            #przesłanie długości słownika i samego słownika
            client_socket.send(dictionary_length.to_bytes(2, "little"))
            client_socket.sendall(dictionary)
            data_length = len(message)
            # przesłanie długości wiadomości i samej wiadomości
```

```
            client_socket.send(data_length.to_bytes(2, "little"))
            client_socket.sendall(message)
            break
        except Exception:
            pass


#funckja odpowiedzialna za odebranie wiadomości
def receive(ip: str, port: int):
    # Tworzenie socketu TCP
    # operującego na adresie IPv4
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Powiązanie adresu ip z socketem i rozpoczęcie nasłuchiwania
    server_socket.bind((ip, port))
    server_socket.listen()
    #zaakceptowanie połączenia od klienta
    (client_connected, client_address) = server_socket.accept()
    # odebranie długości słownika i samego słownika
    dictionary_length = client_connected.recv(2)
    dictionary = client_connected.recv(int.from_bytes(dictionary_length, "little"))
    # odebranie długości wiadomości i samej wiadomości
    data_length = client_connected.recv(2)
    message = client_connected.recv(int.from_bytes(data_length, "little"))

    return message, dictionary
```

**Podsumowanie wnioski.**

**Algorytm statycznego kodowania Huffmana pozwala ograniczyć rozmiar wysyłanej wiadomości. W celu odkodowania wiadomości potrzebna jest informacja o oryginalnym słowniku kodowym. Musi być on zatem przesłany niezaszyfrowany wraz z wiadomością, bądź znany wcześniej odbiorcy.**