Telekomunikacja - laboratorium				Studia stacjonarne - inżynierskie	
Nazwa zadania	**Implementacja algorytmu kodowania Huffmana**				
Dzień	**Poniedziałek**	**Godzina**	**14:00 – 15:30**	**Rok akademicki**	**2020/2021**
Imię i Nazwisko	**Adam Kapuściński 229907**				
Imię i Nazwisko	**Damian Szczeciński 230016**				
Imię i Nazwisko					

Opis programu, rozwiązania problemu.

Celem zadania było opracowanie programu zdolnego do skompresowania, wysłania i rozpakowania pliku przez gniazdo sieciowe. Wymagany algorytm kompresji to algorytm kodowania Huffmana. Algorytm działa w następujący sposób:

- obliczamy ilość wystąpień danego słowa (jednego lub więcej bajtów) w pliku
- tworzymy drzewo kodowe w taki sposób, że liście tego drzewa zawierają słowa, a wszystkie węzły zawierają wagi. Węzły łączymy w taki sposób, że mniejszy jest po lewej, a większy po prawej, natomiast ich rodzicem jest węzeł o wadze sumy wagi dzieci. Każde przejście po drzewie (od korzenia) w lewo to bitowe „0”, a przejście w prawo to bitowe „1”. W ten sposób optymalizujemy ilość zużytych bitów na dane słowo (im słowo występuje częściej tym ma krótszą definicję)
- W naszym programie, dla wygody przepisujemy drzewo na słownik, gdzie kluczem słownika jest słowo, a wartością pod kluczem jest definicja

W przypadku komunikacji sieciowej przyjęliśmy następującą konwencję:

- Serwer wysyła informację o rozmiarze drzewa kodowego
- Serwer wysyła drzewo kodowe
- Serwer kompresuje i wysyła plik

Jako, że klient na początku otrzymuje rozmiar drzewa, to jest w stanie odróżnić gdzie powinien zakończyć odbieranie drzewa, a zacząć odbieranie pliku. Wielkość paczek w jakich transmitowane są te dane jest konfigurowalna w naszym programie, ale domyślnie używamy pakietów o rozmiarze 512B.

Nasz program charakteryzuje się tym, że procesy odczytu pliku z dysku, kompresji i wysyłania są ze sobą ściśle powiązane, dzięki czemu nie zużywamy dużo zasobów, jednak wadą tego rozwiązania jest niezwykle wolna transmisja po sieci.

Najważniejsze elementy kodu programu z opisem.

Funkcje odpowiedzialne za odpowiednio: obliczenie wystąpień słów w pliku, utworzenie drzewa, utworzenie słownika na podstawie drzewa:

```
1.  def __analyze(self):
2.          freq = {}
3.          # zaanalizuj plik bajt po bajcie
4.          while True:
5.              byte = self.FILE.read(WORD_SIZE)
6.              # jeżeli koniec pliku to zakończ
7.              if not byte: break
8.              # dopisz / zapisz ilość wystąpień danego byte'u
9.              if not byte in freq: freq[byte] = 1
10.             else: freq[byte] += 1
11.         # posortuj po ilości wystąpień
12.         freq = dict(sorted(freq.items(), key=lambda item: item[1]))
13.         # wróć na początek pliku
```

```
        self.FILE.seek(0)
        # przepisz słownik frekwencji na listę node'ów
        tmp = []
        i = 0
        for key in freq:
            tmp.append(Node(key, freq[key]))
            tmp[i].leaf = True
            i += 1
        self.tree = tmp

    def __createTree(self):
        if type(self.tree) != list: return
        while len(self.tree) > 1:
            # posortuj listę po freq
            self.tree = sorted(self.tree, key=lambda x : x.freq)

            # weź 2 node'y o najmniejszym freq
            left = self.tree[0]
            right = self.tree[1]
            # utwórz nowy node o wartości freq sumy lewego i prawego
            newNode = Node(b'', left.freq + right.freq, left, right)

            # usuń node'y wykorzystane do zbudowania pod drzewa
            self.tree.remove(left)
            self.tree.remove(right)
            # dodaj poddrzewo do listy
            self.tree.append(newNode)
        # dla wygody przepisz obiekt
        self.tree = self.tree[0]

    def __rewriteAsDict(self, node=None, code=''):
        # jeżeli pierwsze odwołanie, to zacznij od roota
        if node == None: node = self.tree

        # jak idziesz w lewo, to dopisz '0'
        if node.left:
            self.__rewriteAsDict(node.left, code + "0")

        # jak idziesz w prawo, to dopisz '1'
        if node.right:
            self.__rewriteAsDict(node.right, code + "1")

        # jak to liść, to zapisz kod w słowniku
        if node.isLeaf():
            self.dic[node.char] = code
```

odczyt i kompresja pliku:

```
def readNext(self):
        to_send = BitArray()
        # okreslenie ile razy ma się wykonać pętla (ilość bitów)
        bits_to_read = PACKET_SIZE
        while len(to_send.bin) != bits_to_read:
            # jeżeli wartość jest pusta to
            if len(self.buffer) == 0:
                # przypisujemy pierwszy bajt z pliku
                byte = self.FILE.read(WORD_SIZE)
                # jeżeli nie ma to kończymy
                if not byte:
                    self.EOF = True
                    break
                # przypisujemy wartości definicję
                self.buffer = self.dic[byte]
            # dodajemy bit do naszej tablicy
            to_send.bin += self.buffer[0]
            # ucinamy dodany bit
            self.buffer = self.buffer[1:]
          return to_send.tobytes()
```

funkcja dekompresująca i zapisująca dane do pliku

```
def write(self, data):
        # data (bytes) -> BitArray
        data = BitArray(data)
        result = b''
        # przejdź po każdym bicie
        for bit in data.bin:
            # jeżeli napotkany node jest liściem
            if self.buffer.isLeaf():
                # dodaj jego 'char' do wyniku
                result += self.buffer.char
                # wróć do korzenia
                self.buffer = self.root
            # jeżeli bit to '1', przejdź w prawo
            if bit == '1': self.buffer = self.buffer.right
            # jeżeli bit to '0', przejdź w lewo
            if bit == '0': self.buffer = self.buffer.left
        # zapisz bity do pliku
        self.FILE.write(result)
        # jeżeli nie dokończono jeszcze odczytu jakiegoś bajtu
        # to jest to zapisane w bufforze
```

Klient naszego programu (strona odbierająca):

```
print('Nawiązywanie połączenia.')
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    print('Utworzono połączenie z {}:{}'.format(HOST,PORT))
    tree_size = int.from_bytes(s.recv(PACKET_SIZE), "big") # odbierz rozmiar drzewa
    # odbierz drzewo
    data = b''
    while len(data) != tree_size:
        data += s.recv(PACKET_SIZE)
    tree = pickle.loads(data)
    # utwórz obiekt Writera
    writer = HuffmanWriter(filename, tree)

    # odbierz dane
    i = 1
    while data != b'':
        print("Odbieram pakiet nr: {}".format(i), end="\r")
        i += 1
        data = s.recv(PACKET_SIZE)
        TRANSMITTED_DATA_SIZE += len(data)
        writer.write(data)
    print('')
    # zamknij plik
    writer.close()
```

Serwer naszego programu (strona nadająca):

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    print("Serwer uruchomiono na porcie: {}".format(PORT))
    conn, addr = s.accept()
    with conn:
        print('Nawiązano połączenie z: {}'.format(addr))
        conn.sendall(len(tree_str).to_bytes(8, byteorder='big')) # wyślij rozmiar drz
ewa
        # wyślij drzewo
        while len(tree_str) > 0:
            data = tree_str[0:int(PACKET_SIZE/8)]
            conn.sendall(data)
            tree_str = tree_str[int(PACKET_SIZE/8):]
        # wyślij plik
        i = 1
        while not reader.isEOF():
            print("Wysyłam pakiet nr: {}".format(i), end="\r")
            i+=1
            data = reader.readNext()
```

```
            TRANSMITTED_DATA_SIZE += len(data)
            conn.sendall(data)
        print('')
        # zamknij plik
          reader.close()
```

Podsumowanie wnioski.

Cel zadania możemy uznać za wykonany. Program jest w stanie bez problemu nadać i odebrać dowolny plik przez gniazdo sieciowe, gdzie serwer wcześniej kompresuje dane, a klient je dekompresuje.

Nasza metoda odczytywania i kompresji pliku na bieżąco jest bardzo dobra w przypadku transmisji dużych plików które dobrze kompresują się tym algorytmem, czyli np. duże pliki wykonywalne albo bitmapy, jednak dodaje niepotrzebne opóźnienie między wysyłaniem następnych pakietów przy plikach, które z założenia są dobrze skompresowane (mp4, jpg, zip). Można by to poprawić oddelegowując odczyt, kompresję i transmisję na oddzielne wątki które miałyby własne bufory, jednak wymagałoby to przepisania aplikacji z myślą o takiej architekturze.