Sprawozdanie z zajęć 28.03.2023

Konrad Bik

Kwiecień 2023

Spis treści

1 Używane biblioteki

Program został napisany w języku Python w wersji 3.11.2. Do wykonania programu użyłem bibliotek time i math.

2 Program do dekompresji

```
import math, time

if __name__ == '__main__':
    startTime = time.time()
    with open('skompresowany.txt', 'rb') as compressed:
        data = [i for i in compressed.read()]

    uniqueSymbols = data[0]
    dataDict = [chr(data[i]) for i in range(1, uniqueSymbols + 1)]

    toDecomp_L = [bin(data[i])[2:].zfill(8) for i in range(uniqueSymbols + 1, len(data))]
    binText = ''.join(toDecomp_L)

    rest = int(binText[:3], 2)
    toDecomp = binText[3:(len(binText) - rest)]

    N = math.ceil(math.log2(uniqueSymbols))
    decompressed = [dataDict[int(toDecomp[i:(i + N)], 2)] for i in range(0, len(toDecomp), N)]

    with open('zdekompresowany.txt', 'w', encoding='utf-8') as decomp:
        decomp.write(''.join(decompressed))

    endTime = time.time()
    print(f'Elapsed time {endTime - startTime}')
```

Pierwsza linijka importuje biblioteki math i time. Następnie sprawdzamy, czy kod jest uruchamiany jako plik główny programu. W zmiennej startTime zapisujemy czas rozpoczęcia działania programu. Otwieramy plik "skompresowany.txt"w trybie odczytu binarnego, a następnie zapisujemy odczytane dane do listy data. Pierwszy element tej listy reprezentuje liczbę unikalnych symboli użytych w skompresowanym pliku. Tworzymy słownik danych, który zawiera unikalne symbole oraz odpowiadające im wartości liczbowe. W celu zdekompresowania tekstu, kolejne elementy listy data, które reprezentują zakodowane symbole, są przekształcane w ciąg bitów i zapisywane do listy toDecomp_L. Lista toDecomp_L jest łączona w jedną całość, tworząc binarny tekst skompresowanego pliku w postaci zmiennej binText. Pierwsze trzy bity binText reprezentują wartość resztową, którą należy odczytać, aby odzyskać oryginalny tekst. Pozostałe bity binText to zdekodowany tekst po usunięciu wartości reszty. Następnie obliczamy wartość N, która reprezentuje liczbę bitów potrzebną do zapisania liczby unikalnych symboli. Dekodujemy tekst, dzieląc go na podciągi o długości N i zamieniając każdy podciąg na odpowiadający mu symbol z listy dataDict. Ostatecznie dekompresowany tekst zostaje zapisany w pliku "zdekompresowany.txt". W zmiennej endTime zapisujemy czas zakończenia działania programu. Na końcu program wyświetla czas, jaki upłynął podczas dekompresji pliku.

3 Program do kompresji z szyfrowaniem

```
import math, time

def decBin(rest, x):
    binary = str(bin(rest)[2:])
    return binary.rjust(x, '0')
```

Funkcja konwertuje liczbę dziesiętną na liczbę binarną i uzupełnia brakujące zera przed liczbą, aby uzyskać długość x.

```
def getFromFile():
    with open('do_kompresji.txt', 'r', encoding='utf-8') as file:
        data = ''.join(file.read())
    return data
```

Funkcja odczytuje zawartość pliku "do_kompresji.txt"i zwraca ją jako ciąg znaków.

```
def encrypt():
    asciiTab = [[col + row - 256 if col + row > 255 else col + row
                 for col in range(256)] for row in range(256)]
```

```
    encTab = []
    index = 0

    for char in res:
        keyChar = ord(key[index])
        encTab.append(asciiTab[keyChar][char])
        index = (index + 1) % len(key)

    return encTab
```

Funkcja wykonuje szyfrowanie XOR na danych wejściowych z kluczem.

```
if __name__ == '__main__':
    key = input('Key: ')
    if len(key) == 0:
        print('No key')
        exit()

    dataBin = ''
    data = getFromFile()
    dataDict = sorted(list(set(data)))

    X = len(dataDict)
    N = math.ceil(math.log2(X))
    R = (8 - (3 + N * len(data))%8)%8

    print(f'X = {X}\nN = {N}\nR = {R}')
    print(f'File lenght before compression: {len(data)}')
    startTime = time.time()

    res = bytearray()
    res.append(X)

    for j in dataDict:
        res.append(ord(j))

    dataBin = decBin(R, 3)
    binSecond = ''

    for char in data:
        binSecond += decBin(dataDict.index(char), N)

    binSecond += str(1) * R
    dataBin += binSecond

    for i in range(0, len(dataBin), 8):
        swToChar = chr(int(dataBin[i:(i+8)], 2))
        res.append(ord(swToChar))

    encTab = encrypt()

    with open('skompresowany_e.txt', 'wb') as comp:
        comp.write(bytes(encTab))

    endTime = time.time()
    print(f'File lenght after compression: {len(encTab)}')
    print(f'Elapsed time {endTime - startTime}')
```

Program wczytuje plik "do_kompresji.txt"i kompresuje go przy użyciu algorytmu LZW. Skompresowane dane są następnie szyfrowane za pomocą operacji XOR z kluczem, który użytkownik podaje na początku programu. Na początku programu użytkownik podaje klucz, który jest wykorzystywany do szyfrowania danych wejściowych. Następnie dane wejściowe są wczytywane z pliku "do_kompresji.txt". Algorytm LZW jest następnie wykorzystywany do kompresji danych. Kompresja odbywa się za pomocą słownika, w którym każdy unikalny ciąg znaków jest przyporządkowany unikalnemu kodowi numerycznemu. Kompresja jest przeprowadzana w taki sposób, że każdy ciąg znaków jest zastępowany odpowiadającym mu kodem numerycznym. Znaki i odpowiadające im kody są następnie zapisywane w pliku "skompresowany_e.txt". W kolejnym kroku dane skompresowane są szyfrowane za pomocą operacji XOR z kluczem podanym na początku programu. Zaszyfrowane dane są zapisywane w pliku "skompresowany_e.txt". Program kończy się wyświetleniem długości danych wejściowych i skompresowanych, a także czasu wykonania całego programu.

4 Program do dekompresji z szyfrowaniem

```
import math, time

def decrypt(data):
    key = input('Key: ')
    if len(key) == 0:
        print('No key')
        exit()

    asciiTab = [[col + row - 256 if col + row > 255 else col + row
                 for col in range(256)] for row in range(256)]

    decTab = []
    index = 0

    for char in data:
        keyChar = ord(key[index])
        decTab.append(asciiTab[keyChar].index(char))
        index = (index + 1) % len(key)
    return decTab
```

Funkcja decrypt(data) otrzymuje na wejściu zaszyfrowane dane w postaci listy bajtów. Na początku pobiera od użytkownika klucz, który będzie używany do odszyfrowania danych. Jeśli długość klucza jest równa 0, program wypisuje informację o braku klucza i kończy działanie. Następnie funkcja tworzy tablicę ASCII asciiTab, w której każdy element jest sumą indeksów wiersza i kolumny, pomniejszoną o 256, jeśli suma ta jest większa od 255. Powstała w ten sposób tablica jest używana do odszyfrowania poszczególnych znaków zaszyfrowanych danych. W pętli for funkcja iteruje po zaszyfrowanych danych, pobierając kolejne znaki i indeksując tablicę asciiTab przy pomocy znaku klucza o odpowiednim indeksie. Zwrócone indeksy są zapisywane do listy decTab, a zmienna index jest aktualizowana, aby wskazywać na kolejny znak klucza.

```
if __name__ == '__main__':
    startTime = time.time()
    with open('skompresowany_e.txt', 'rb') as compressed:
        data_e = [i for i in compressed.read()]

    data = decrypt(data_e)
    uniqueSymbols = data[0]
    dataDict = [chr(data[i]) for i in range(1, uniqueSymbols + 1)]
    toDecomp_L = [bin(data[i])[2:].zfill(8) for i in range(uniqueSymbols + 1, len(data))]

    binText = ''.join(toDecomp_L)
    rest = int(binText[:3], 2)
    toDecomp = binText[3:(len(binText) - rest)]

    N = math.ceil(math.log2(uniqueSymbols))
    decompressed = [dataDict[int(toDecomp[i:(i + N)], 2)] for i in range(0, len(toDecomp), N)]

    with open('zdekompresowany_e.txt', 'w', encoding='utf-8') as decomp:
        decomp.write(''.join(decompressed))

    endTime = time.time()
    print(f'Elapsed time {endTime - startTime}')
```

Główna część programu zaczyna się od wczytania zaszyfrowanych danych z pliku "skompresowany_e.txt"i przekazania ich do funkcji decrypt. Wynik odszyfrowania jest zapisywany w zmiennej data, a następnie jest ona przetwarzana, aby odzyskać oryginalne dane. W pierwszym kroku program odczytuje liczbę unikalnych symboli, która jest zapisana na początku zaszyfrowanych danych, i tworzy listę dataDict, zawierającą odpowiadające im znaki. Kolejnym krokiem jest przetworzenie zaszyfrowanych danych na ciąg binarny, który jest zapisywany w zmiennej binText. Pierwsze trzy bity binarne tej zmiennej określają liczbę bitów, które zostały użyte do zapisania ostatniego symbolu, a te bity są zapisywane w zmiennej rest. Następnie program dzieli binText na fragmenty o długości N, gdzie N to najmniejsza liczba bitów potrzebna do zapisania liczby unikalnych symboli, i przekształca każdy fragment w odpowiadający mu symbol zgodnie z wartościami zapisanymi w dataDict. Wynik jest zapisywany do zmiennej decompressed. Na koniec zdekompresowane dane są zapisywane do pliku "zdekompresowany_e.txt", a czas trwania działania programu jest wyświetlany na ekranie.