Akademia Nauk Stosowanych - Teoretyczne i technologiczne podstawy multimediów - laboratorium			
Temat: Algorytm Shannona – Fano.			Symbol: TiTPM
Nazwisko i imię: Fyda Kamil		Ocena sprawozdania	Zaliczenie:
Data wykonania ćwiczenia: 25.10.2022r.	Grupa: L1		

1. Opis teoretyczny:

Kodowanie Shannona-Fano – metoda kompresji bezstratnej autorstwa Roberta Fano. Kodowanie to dla dyskretnego źródła danych znajduje kod prefiksowy, który charakteryzuje się dość dobrą efektywnością – lepszą od kodowania Shannona (słowa kodowe krótsze o 1 bit), nie tworzy jednak optymalnych kodów. Kodowanie Shannona-Fano jest używane w kompresorze ZIP, przy wybranej metodzie kompresji implode.

2. Algorytm tworzenia słów kodowych:

Algorytm przedstawia się następująco:

- s – ciąg symboli ze zbioru S posortowanych według prawdopodobieństw pi.

Shannon-Fano(s):

- Jeśli **s** zawiera dwa symbole, do słowa kodu pierwszej litery dodaje **0**, do słowa kodu drugiej litery – **1**.

- W przeciwnym razie, jeśli **s** zawiera więcej niż dwa symbole, podzielono go na dwa podciągi **s1** i **s2** tak, żeby różnica między sumą prawdopodobieństw liter z **s1** i **s2** była najmniejsza. Do słów kodu symboli z **s1** można dodać **0**, do kodów symboli z **s2** – **1**. Wywołanie rekurencyjne funkcji: Shannon-Fano(**s1**) oraz Shannon-Fano(**s2**).

3. Algorytm kodowania Shannona-Fano – krok po kroku:

1. Określenie prawdopodobieństwa wystąpienia wszystkich symboli (waga symboli).
2. Sortowanie listy symboli według prawdopodobieństwa.
3. Ustalenie posortowanej listy jako zbiór główny.
4. Podział grupy symboli na dwie części o możliwie równej sumie wadze symboli.
5. Przyporządkowanie symbolom z jednej grupy binarne „0”, zaś symbolom z drugiej grupy binarne „1”
6. Powtórzenie dla każdej podgrupy od punktu 4, aż do uzyskania podgrupy złożonej z jednego symbolu
7. Przyporządkowanie kolejnym symbolom z listy słowa kodowe, składające się z bitów kolejno przyporządkowanym grupom, do których trafiał symbol w kolejnych podziałach.

4. Kod programu:

> Users > Asus > Desktop > kodowanie.py.txt > krawedzie_interw

```python
import numpy as np
def skaluj_kraw_interw(orgin_kraw_interw, min_val, max_val):
    nowe_krawedzie = min_val + (max_val - min_val)*orgin_kraw_interw
    return nowe_krawedzie
def symbol_na_indeks(symbol, alfabet):
    assert len(set(alfabet)) == len(alfabet), 'Niepotrzebny'
    assert symbol in alfabet, 'Symbolu {} nie ma w alfabecie'.format(symbol)
    return alfabet.index(symbol) + 1 #


def pobierz_inter_z_symbolu(aktualny_symbol, alfabet, aktualny_min, aktualny_max, orgin_kraw_interw):
    """Uzyskujemy nowy interwał dla nowego symbolu na podstawie bieżących min i maks oraz oryginalnych krawędzi interwału"""
    aktualny_sygnal_ind = symbol_na_indeks(aktualny_symbol, alfabet)
    aktualne_krawedzie_interw = skaluj_kraw_interw(orgin_kraw_interw, aktualny_min, aktualny_max)
    nowe_min = aktualne_krawedzie_interw[aktualny_sygnal_ind - 1]
    nowe_max = aktualne_krawedzie_interw[aktualny_sygnal_ind]

    return (nowe_min, nowe_max)
def krawedzie_interw(pmf):
    """Zwracamy listę krawędzi przedziałów, biorąc pod uwagę funkcję masy prawdopodobieństwa"""
    return np.array([np.sum(pmf[:i]) for i in range(len(pmf) + 1)])


def interw_arytmet(alfabet, sygnal, pmf):
    """Uzyskujemy listę przedziałów uzyskanych przez arytmetyczne kodowanie sygnału.


    Zwraca:
        Przedziały: Lista (min, max) krotek zmiennoprzecinkowych,
        gdzie (min, max) to krawędzie przedziału odpowiedniego symbolu. Ta lista ma jeden przedział na symbol
        w sygnał wejściowy..
    """
    orgin_kraw_interw = krawedzie_interw(pmf)
    sygnal_list = list(sygnal)
    aktualny_min, aktualny_max = pobierz_inter_z_symbolu(sygnal_list[0], alfabet, 0.0, 1.0, orgin_kraw_interw)
    przedzialy = [(aktualny_min, aktualny_max)]
    for i, symbol in enumerate(sygnal_list[1:]):
        aktualny_min, aktualny_max = pobierz_inter_z_symbolu(symbol, alfabet, aktualny_min, aktualny_max, orgin_kraw_interw)
        przedzialy.append((aktualny_min, aktualny_max))
    return przedzialy

#na binarne
def sekwencja_binarna(dziesietne):
    """Zwraca binarną reprezentację danych wejściowych.

    Argumenty, jakie przyjmuje:
        dziesietne: skalarne float w [0, 1) przedział jest w połowie otwarty
    Zwraca:
        sekwencja_bin: Lista liczb całkowitych w {0, 1}
    """
    reminder = dziesietne
    sekwencja_bin = [int(np.floor(2 * reminder))]
    ind = 0
    while reminder > 1e-10:
        reminder = 2 * reminder - sekwencja_bin[ind]
        sekwencja_bin.append(int(np.floor(2 * reminder)))
        ind += 1
    return sekwencja_bin

#najkrotsze
def najkrotsza_binarnie(d_interval):
    d_min, d_max = d_interval
    assert d_min < d_max, 'Potrzebujemy ściśle rosnącego interwału'
    assert d_min >= 0, 'Ujemna dolna granica na interwale / przedziale'
    assert d_max < 1, 'Górna granica przedziału większa lub równa 1'

    c_min = 0.0
    c_max = 1.0
    k = 1
    bin_sekw = []
    while True:
```

```
            if c_min < d_min and c_min + 1 / 2 ** k < d_max:
                c_min = c_min + 1 / 2 ** k
                bin_sekw.append(1)
            else:
                if c_max > d_max and c_max - 1 / 2 ** k > d_min:
                    c_max = c_max - 1 / 2 ** k
                    bin_sekw.append(0)
                else:
                    break
            k = k + 1
        return bin_sekw

    #kodowanie
    def arithmetic_encoding(alfabet, pmf, sygnal):

        przedzialy = interw_arytmet(alfabet, sygnal, pmf)
        bin_sekw = najkrotsza_binarnie(przedzialy[-1])
        return bin_sekw

    #dekodowanie
    def interval_w_narastaniu(numer, lista_wzrast):
        """Znajdujemy przedział, w którym liczba znajduje się na liście rosnących liczb."""
        assert numer >= lista_wzrast[0], 'numer jest poza lista_wzrast'
        assert numer <= lista_wzrast[-1], 'numer jest poza lista_wzrast'
        interval = None
        for i in range(1, len(lista_wzrast)):
            if numer >= lista_wzrast[i-1] and numer <= lista_wzrast[i]:
                interval = (i-1, i)
        assert interval is not None, 'Nieprawidłowe dane wejściowe do interval_w_narastaniu'
        return interval


    def dziesietne_z_binarnych(bin_sekw):

        half_powered = np.array([1 / 2 ** k for k in range(1, len(bin_sekw) + 1)])
        return np.dot(np.array(bin_sekw), half_powered)
def dekodowanie_arytm(alfabet, pmf, zakodowany_sygnal, num_to_decode):

    # Inicjalizacja
    dziesietne_sygnal = dziesietne_z_binarnych(zakodowany_sygnal)
    orgin_kraw_interw = krawedzie_interw(pmf)
    aktual_symbol_inds = interval_w_narastaniu(dziesietne_sygnal, orgin_kraw_interw)
    symbole = [alfabet[aktual_symbol_inds[0]]]
    nowe_min = orgin_kraw_interw[aktual_symbol_inds[0]]
    nowe_max = orgin_kraw_interw[aktual_symbol_inds[1]]
    for num_decoded in range(1, num_to_decode):
        aktualne_krawedzie_interw = skaluj_kraw_interw(orgin_kraw_interw, nowe_min, nowe_max)
        aktual_symbol_inds = interval_w_narastaniu(dziesietne_sygnal, aktualne_krawedzie_interw)
        symbole.append(alfabet[aktual_symbol_inds[0]])
        nowe_min = aktualne_krawedzie_interw[aktual_symbol_inds[0]]
        nowe_max = aktualne_krawedzie_interw[aktual_symbol_inds[1]]

    return ''.join(symbole)
alfabet = ['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']
pmf = np.array([8.167, 1.492, 2.782, 4.253, 12.702, 2.228, 2.015, 6.094, 6.966, 0.153, 0.772, 4.025, 2.406, 6.749, 7.507, 1.929, 0.095, 5.987, 6.327, 9.056, 2.758, 0.978,
sygnal = input("Podaj tekst do zakodowania: ")
all_freq = {}

for i in sygnal:
    if i in all_freq:
        all_freq[i] += 1
    else:
        all_freq[i] = 1



zakodowany_sygnal = arithmetic_encoding(alfabet, pmf, sygnal)
odkodowany_sygnal = dekodowanie_arytm(alfabet, pmf, zakodowany_sygnal, len(sygnal))

print('Tekst, ktory podales:         {}'.format(sygnal))
print('Zakodowany tekst: {}'.format(''.join([str(s) for s in zakodowany_sygnal])))
print('Odkodowany tekst: {}'.format(odkodowany_sygnal))
print("Liczba danych znakow w podanym przez Nas tekscie liczy sobie :\n "+ str(all_freq))
```

5. Wynik działania:

```
PS C:\Users\Asus> & C:/Users/Asus/AppData/Local/Programs/Python/Python310/python.exe c:/Users/Asus/Desktop/kodowanie.py.txt
Podaj tekst do zakodowania: alamakota
Tekst, ktory podales:         alamakota
Zakodowany tekst: 0000100111111110000110000100111010011110
Odkodowany tekst: alamakota
Liczba danych znakow w podanym przez Nas tekscie liczy sobie :
 {'a': 4, 'l': 1, 'm': 1, 'k': 1, 'o': 1, 't': 1}
```

```
PS C:\Users\Asus> & C:/Users/Asus/AppData/Local/Programs/Python/Python310/python.exe c:/Users/Asus/Desktop/kodowanie.py.txt
Podaj tekst do zakodowania: kamilfyda
Tekst, ktory podales:         kamilfyda
Zakodowany tekst: 011110000000011010110010011010100001010100000
Odkodowany tekst: kamilfyda
Liczba danych znakow w podanym przez Nas tekscie liczy sobie :
 {'k': 1, 'a': 2, 'm': 1, 'i': 1, 'l': 1, 'f': 1, 'y': 1, 'd': 1}
```

```
PS C:\Users\Asus> & C:/Users/Asus/AppData/Local/Programs/Python/Python310/python.exe c:/Users/Asus/Desktop/kodowanie.py.txt
Podaj tekst do zakodowania: zielonatrawa
Tekst, ktory podales:         zielonatrawa
Zakodowany tekst: 11111111111000101110010110001110100110001101111011010
Odkodowany tekst: zielonatrawa
Liczba danych znakow w podanym przez Nas tekscie liczy sobie :
 {'z': 1, 'i': 1, 'e': 1, 'l': 1, 'o': 1, 'n': 1, 'a': 3, 't': 1, 'r': 1, 'w': 1}
```