

Akademia Nauk Stosowanych w Nowym Sączu			
Teoretyczne i technologiczne podstawy multimediów– laboratorium			
Temat: Teoretyczne i technologiczne podstawy multimediów.			L2
Nazwisko i imię: Szczepanek Piotr		Ocena sprawozdania	Zaliczenie:
Data wykonania ćwiczenia: 18.10.2022		Grupa: L2	

Kodowanie Fano - Shannona – kodowanie polegające na zastępowaniu symboli w łańcuchu wejściowym sekwencjami bitów. Idea ta jest identyczna jak ta przyświecająca kodowaniu Huffmana.

Najważniejsze aspekty to:

- Najpopularniejszy znak otrzymuje najkrótszy kod
- Kod dowolnego znaku nie przedrostkiem kody innego znaku

Przebieg algorytmu:

- 1) Na samym początku zliczamy wystąpienia każdego ze znaków:
W Naszym przypadku, będziemy sprawdzać wyraz TESTOWE
- 2) Teraz sortujemy wystąpienia względem ich częstotliwości od najpopularniejszego do najmniej popularnego.
- 3) Tworzymy listę symboli na bazie posortowanego zbioru.
- 4) Dokonujemy podziału zbioru na dwie części tak by suma częstości była możliwie jak najbardziej równa.
- 5) Zbiorowi po lewej stronie przypisujemy bit 0, a zbiorowi po prawej bit 1.
- 6) Dokonujemy podziału obu zbiorów na dwie połowy znów tak by sumy wag były jak najbardziej równe.
- 7) Ponownie przypisujemy lewemu podzbiorowi bit 0, a prawemu bit 1.
- 8) Podziałów dokonujemy do momentu, aż nie będzie już co dzielić.
- 9) Odczytujemy kody.
- 10) Na koniec odczytujemy wyniki.

```

1 import numpy as np
2 def skaluj_kraw_interw(organ_kraw_interw, min_val, max_val):
3     """Skalujemy oryginalne krawędzie przedziału do nowych wartości minimalnych i maksymalnych"""
4     nowe_krawedzie = min_val + (max_val - min_val) * organ_kraw_interw
5     return nowe_krawedzie
6 def symbol_na_indeks(symbol, alfabet):
7     """Przyporządkuj symbol do indeksu według alfabetu (indeksowanie na podstawie 1, ponieważ pracujemy na przedziałach,
8     zwracany indeks będzie zgodny z górną krawędzią interwału.)"""
9     assert len(set(alfabet)) == len(alfabet), 'Niepotrzebny'
10    assert symbol in alfabet, 'Symbolu {} nie ma w alfabecie'.format(symbol)
11    return alfabet.index(symbol) + 1 #
12
13
14 def pobierz_inter_z_symbolu(aktualny_symbol, alfabet, aktualny_min, aktualny_max, organ_kraw_interw):
15     """Uzyskujemy nowy interwał dla nowego symbolu na podstawie bieżących min i maks oraz oryginalnych krawędzi interwału"""
16     aktualny_sygnal_ind = symbol_na_indeks(aktualny_symbol, alfabet)
17     aktualne_krawedzie_interw = skaluj_kraw_interw(organ_kraw_interw, aktualny_min, aktualny_max)
18     nowe_min = aktualne_krawedzie_interw[aktualny_sygnal_ind - 1]
19     nowe_max = aktualne_krawedzie_interw[aktualny_sygnal_ind]
20
21     return (nowe_min, nowe_max)
22 def krawedzie_interw(pmf):
23     """Zwracamy listę krawędzi przedziałów, biorąc pod uwagę funkcję masy prawdopodobieństwa"""
24     return np.array([np.sum(pmf[:i]) for i in range(len(pmf) + 1)])
25
26
27 def interw_arytmety(alfabet, sygnal, pmf):
28     """Uzyskujemy listę przedziałów uzyskanych przez arytmetyczne kodowanie sygnału.
29
30     Arg:
31         alfabet: Lista unikalnych znaków.
32         sygnal: Ciąg symboli, w którym każdy symbol musi należeć do alfabetu.
33         pmf: Funkcja masy prawdopodobieństwa dla elementów w alfabecie.
34             pmf[i] jest prawdopodobieństwem alfabetu[i].
35
36     Zwraca:
37         Przedziały: Lista (min, max) krotek zmiennoprzecinkowych,
38         gdzie (min, max) to krawędzie przedziału odpowiedniego symbolu. Ta lista ma jeden przedział na symbol
39         w sygnale wejściowym..
40     """
41     organ_kraw_interw = krawedzie_interw(pmf)
42     sygnal_list = list(sygnal)
43     aktualny_min, aktualny_max = pobierz_inter_z_symbolu(sygnal_list[0], alfabet, 0.0, 1.0, organ_kraw_interw)
44     przedzialy = [(aktualny_min, aktualny_max)]
45
46     for i, symbol in enumerate(sygnal_list[1:]):
47         aktualny_min, aktualny_max = pobierz_inter_z_symbolu(symbol, alfabet, aktualny_min, aktualny_max, organ_kraw_interw)
48         przedzialy.append((aktualny_min, aktualny_max))
49     return przedzialy
50
51 #na binarne
52 def sekwencja_binarna(dziesietne):
53     """Zwraca binarną reprezentację danych wejściowych.
54
55     Argumenty, jakie przyjmuje:
56         dziesiętne: skalarne float w [0, 1) przedział jest w połowie otwarty
57
58     Zwraca:
59         sekwencja_bin: Lista liczb całkowitych w {0, 1}
60     """
61     reminder = dziesiętne
62     sekwencja_bin = [int(np.floor(2 * reminder))]
63     ind = 0
64     while reminder > 1e-10:
65         reminder = 2 * reminder - sekwencja_bin[ind]
66         sekwencja_bin.append(int(np.floor(2 * reminder)))
67         ind += 1
68     return sekwencja_bin
69
70 #najkrótsze
71 def najkrótsza_binarnie(d_interval):
72     """Znajdujemy najkrótszą reprezentację binarną w przedziale dziesiętnym."""
73     d_min, d_max = d_interval
74     assert d_min < d_max, 'Potrzebujemy ściśle rosnącego interwału'
75     assert d_min >= 0, 'Ujemna dolna granica na interwale / przedziale'
76     assert d_max < 1, 'Górna granica przedziału większa lub równa 1'
77
78     # Inicjalizacja bieżącego przedziału (c_min, c_max)
79     c_min = 0.0
80     c_max = 1.0
81
82     # Kontynuujemy, aż obecny przedział (c_min, c_max) będzie zawarty w środku
83     # przedziału (d_min, d_max).
84     # Dodanie 1 do sekwencji binarnej zwiększa c_min, ale pozostawia c_max stałe.
85     # Dodanie 0 do sekwencji binarnej zmniejsza c_max, ale pozostawia c_min stałe.

```

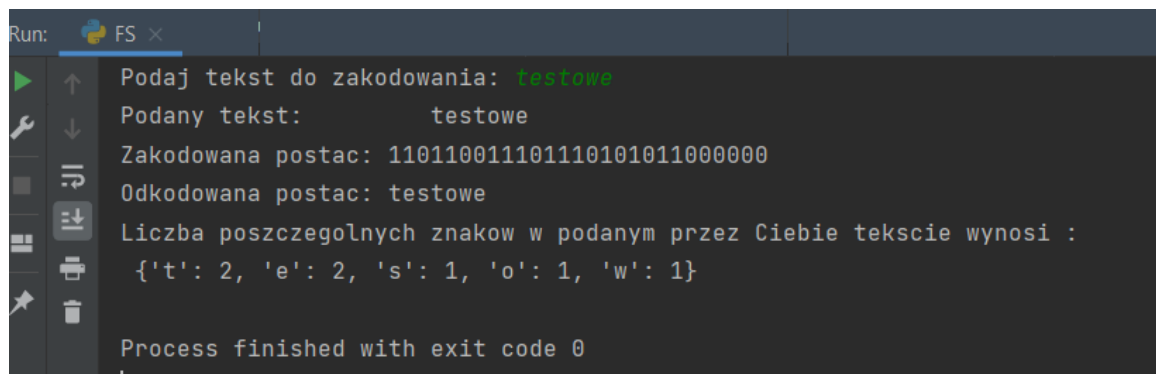
```

86     k = 1
87     bin_sekw = []
88     while True:
89         # Zawsze dodaje 1 do bin_sekw, jeśli to możliwe
90         if c_min < d_min and c_min + 1 / 2 ** k < d_max:
91             c_min = c_min + 1 / 2 ** k
92             bin_sekw.append(1)
93         else:
94             if c_max > d_max and c_max - 1 / 2 ** k > d_min:
95                 c_max = c_max - 1 / 2 ** k
96                 bin_sekw.append(0)
97             else:
98                 # Brak zmian w bieżącym interwale, skończyliśmy
99                 break
100         k = k + 1
101     return bin_sekw
102
103 #kodowanie
104 def arithmetic_encoding(alfabet, pmf, sygnał):
105
106     przedziały = interw_arytm(alfabet, sygnał, pmf)
107     bin_sekw = najkrotsza_binarnie(przedziały[-1])
108     return bin_sekw
109
110 #dekodowanie
111 def interval_w_narastaniu(numer, lista_wzrost):
112     """Znajdujemy przedział, w którym liczba znajduje się na liście rosnących liczb."""
113     assert numer >= lista_wzrost[0], 'numer jest poza lista_wzrost'
114     assert numer <= lista_wzrost[-1], 'numer jest poza lista_wzrost'
115     interval = None
116     for i in range(1, len(lista_wzrost)):
117         if numer >= lista_wzrost[i-1] and numer <= lista_wzrost[i]:
118             interval = (i-1, i)
119     assert interval is not None, 'Nieprawidłowe dane wejściowe do interval_w_narastaniu'
120     return interval
121
122
123 def dziesiętne_z_binarnych(bin_sekw):
124
125     half_powered = np.array([1 / 2 ** k for k in range(1, len(bin_sekw) + 1)])
126     return np.dot(np.array(bin_sekw), half_powered)
127
128
129 def dekodowanie_arytm(alfabet, pmf, zakodowany_sygnał, num_to_decode):
130
131     # Inicjalizacja
132     dziesiętne_sygnał = dziesiętne_z_binarnych(zakodowany_sygnał)
133     orgin_kraw_intew = krawedzie_intew(pmf)
134     aktual_symbol_inds = interval_w_narastaniu(dziesiętne_sygnał, orgin_kraw_intew)
135     symbole = [alfabet[aktual_symbol_inds[0]]]
136     nowe_min = orgin_kraw_intew[aktual_symbol_inds[0]]
137     nowe_max = orgin_kraw_intew[aktual_symbol_inds[1]]
138     for num_decoded in range(1, num_to_decode):
139         aktualne_krawedzie_intew = skaluj_kraw_intew(orgin_kraw_intew, nowe_min, nowe_max)
140         aktual_symbol_inds = interval_w_narastaniu(dziesiętne_sygnał, aktualne_krawedzie_intew)
141         symbole.append(alfabet[aktual_symbol_inds[0]])
142         nowe_min = aktualne_krawedzie_intew[aktual_symbol_inds[0]]
143         nowe_max = aktualne_krawedzie_intew[aktual_symbol_inds[1]]
144
145     return ''.join(symbole)
146
147 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']
148 pmf = np.array([8.167, 1.492, 2.782, 4.253, 12.782, 2.228, 2.015, 6.894, 6.966, 0.153, 0.772, 4.025, 2.406, 6.749, 7.507, 1.929, 0.0])
149 sygnał = input("Podaj tekst do zakodowania: ")
150 all_freq = {}
151
152 for i in sygnał:
153     if i in all_freq:
154         all_freq[i] += 1
155     else:
156         all_freq[i] = 1
157
158 zakodowany_sygnał = arithmetic_encoding(alfabet, pmf, sygnał)
159 odkodowany_sygnał = dekodowanie_arytm(alfabet, pmf, zakodowany_sygnał, len(sygnał))
160
161 print('Podany tekst: {}'.format(sygnał))
162 print('Zakodowana postać: {}'.format(''.join([str(s) for s in zakodowany_sygnał])))
163 print('Odkodowana postać: {}'.format(odkodowany_sygnał))
164 print("Liczba poszczególnych znaków w podanym przez Ciebie tekście wynosi : \n "
165       + str(all_freq))

```

Rysunek 1 Kod programu.

Test działania programu



```
Run: FS x
Podaj tekst do zakodowania: testowe
Podany tekst:      testowe
Zakodowana postac: 110110011101110101011000000
Odkodowana postac: testowe
Liczba poszczegolnych znakow w podanym przez Ciebie tekście wynosi :
{'t': 2, 'e': 2, 's': 1, 'o': 1, 'w': 1}

Process finished with exit code 0
```

Rysunek 2 Działanie programu

WNIOSKI

Jak widzimy, Nasz program działa zgodnie z założeniami. Prosi on Użytkownika o podanie tekstu do zakodowania, po czym wyświetla go w konsoli. Następnie przystępuje do kodowania podanej treści oraz ponownie jej wyświetlenia, po czym widzimy odkodowaną postać, identyczną jak tekst, który podał Użytkownik podczas startu Naszego programu.

Na samym końcu widzimy częstotliwość poszczególnych znaków. W przypadku tekstu, który podaliśmy: TESTOWE, możemy łatwo policzyć, że program dokonuje obliczeń w sposób właściwy, ponieważ litera t – występuje dwa razy, na pierwszym i czwartym miejscu, litera e – dwa razy, na drugim i ostatnim miejscu, litera s – jeden raz, na trzecim miejscu, litera o – jeden raz, na piątym miejscu oraz litera w – jeden raz, na szóstym miejscu.