

# Modulacja i kodowanie

## Labolatorium

### Kodowanie źródłowe – Kod Huffman’a

W tym ćwiczeniu zajmiemy się kodowaniem źródłowym (*source coding*).

#### 1. Kodowanie źródłowe

Głównym celem kodowania źródłowego jest zmniejszenie rozmiaru danych do minimalnego poziomu. Najprostsze techniki kodowania źródłowego dotyczą dyskretnych źródeł i po prostu obejmują reprezentowanie każdego kolejnego symbolu źródła przez sekwencję cyfr binarnych. Dla przykładu 27-symbolowy alfabet angielski (łącznie ze znakiem ‘spacji’) może być zakodowany w postaci 5-bitowego bloku.

W większości przypadków dane wejściowe charakteryzują się nie-zerową redundancją. A co oznacza termin „redundancja danych” ?

**Redundancja** (łac. *redundantia* – 'powódź', 'nadmiar', 'zbytek') – nadmiarowość w stosunku do tego, co konieczne lub zwykłe. Redundancja danych oznacza nadmiarowość w reprezentacji danych. Jest to stosunek aktualnej formy prezentacji do jej minimalnej postaci, ale takiej, która pozwala na zawarcie w niej bezstratnie tej informacji.

Zatem, można powiedzieć, że obniżenie redundancji danych jest równoważne bezstratnej kompresji. Z tego powodu kodowanie kanałowe jest często utożsamiane z kompresją bezstratną. W przypadku kodowania kanałowego (ale także kompresji bezstratnej) kompresujemy jedynie formę informacji, a nie ją samą!

##### 1.1 Kodowanie metodą Huffmana.

**Kod Huffmana** jest szczególnym rodzajem optymalnego kodu prefiksowego, który jest powszechnie używany do bezstratnej kompresji danych. Proces wyszukiwania i / lub używania takiego kodu przebiega za pomocą kodowania Huffmana, algorytmu opracowanego przez Davida A. Huffmana, gdy był Sc.D. student w MIT i opublikowany w artykule z 1952 r. "*A Method for the Construction of Minimum-Redundancy Codes*".

(Algorytm kodowania Huffmana przedstawiony został dokładniej w prezentacji wykładowej)

## 1.2 Kodowanie strumienia tekstu metoda Huffmana. (Część obowiązkowa)

Ćwiczenie powinno być wykonane w środowisku Matlab (minimum Matlab 2006a).

- Na początek trzeba zdefiniować alfabet (zbiór symboli):

```
% define alphabet
alphabet =
{'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','r','s','t','
u','v','y','w','z','x','0','1','2','3','4','5','6','7','8','9',' '};
```

- W następnym kroku generujemy losowy tekst o zadanej długości...

```
l=100; % length of text stream
a = 1; % min (first position at our alphabet)
b = size(alphabet,2); % max (last position at our alphabet)
r = round((b-a).*rand(1,1) + a);
rText = '';

for i = 1 : l
    rText = strcat(rText,alphabet{r(i)});
end
```

pytanie: jakie jest prawdopodobieństwo wylosowania każdego symbolu ?

```
% calculate probability for each symbol

alphabetLength = size(alphabet,2);
textLength = size(rText,2);
probabilityOfSymbol = zeros(alphabetLength,1);
for i = 1 : alphabetLength
    for j=1 : textLength
        if rText(j) == alphabet{i}
            probabilityOfSymbol(i) = probabilityOfSymbol(i) + 1;
        end
    end
end
probabilityOfSymbol = probabilityOfSymbol ./ textLength;

% check propabilityOfSymbol
sum(probabilityOfSymbol)
```

- Generujemy słownik Huffmana:

```
% generate a Huffman dictionary

[dict,avglen] = huffmandict(alphabet,probabilityOfSymbol);
```

Każdy znak alfabetu ma teraz przyporządkowany symbol kodowy.

Każdy symbol kodowy składa się z pewnego zbioru 0 i 1. Dlaczego długość symboli kodowych nie jest taka sama ?

- Definiujemy funkcję, która ma zadanie zwrócić symbol kodowy dla zadanego znaku alfabetu:

```
% function return a code symbol (must be defined as a single function)

function codeSymbol = getSymbol(symbol, dict)
    codeSymbol = 'none';
    for i = 1 : size(dict,1)
        if strcmp(char(symbol),char(dict{i,1}))
            codeSymbol = dict{i,2};
            break;
        end
    end
end
```

- Kodujemy wygenerowany wcześniej strumień tekstowy za pomocą kodów Huffmana:

```
% encode on Huffman Code

alphabetLength = size(alphabet,2);
textLength = size(rText,2);
probabilityOfSymbol = zeros(alphabetLength,1);
encodedData = zeros(1);
for j=1 : textLength
    codeSymbol = getSymbol(char(rText(j)), dict);
    disp(codeSymbol);
    actualPosition = size(encodedData,2);
    if (actualPosition == 1)
        actualPosition = 0;
    end
    encodedData( (actualPosition+1):(actualPosition + size(codeSymbol,2))
) = codeSymbol(:)';
end
```

- Porównujemy ‘nasz’ kod z kodem wygenerowanym za pomocą funkcji Matlab:

```
% Huffman encoder from Matlab

comp = huffmanenco(rText,dict);

% compare with Matlab code

isequal(comp,encodedData)
```

- Obliczamy długość oryginalnego (nieskompresowanego) tekstu (reprezentowanych jako 5bitowe bloki)...

```
encodedDataSize = size(encodedData,2);
uncompressedTextLength = 0;
textLength = size(rText,2);

for i = 1 : textLength
    uncompressedTextLength = uncompressedTextLength +
size(dec2bin(char(rText(j))),2);
end
```

- Teraz możemy porównać długość strumieni: oryginalnego (nieskompresowanego) i skompresowanego

```
message = strcat(' Uncompressed Data size (in Bytes):
',num2str(uncompressedTextLength),' b');
disp(message);
message = strcat('Compressed (by huffman method) Data size (in Bytes):
',num2str(encodedDataSize),' b');
disp(message);

compressionRatio = uncompressedTextLength / encodedDataSize;

message = strcat('CompresionRatio : ',num2str(compressionRatio));
disp(message);
```

- Porównaj współczynnik kompresji dla różnych długości strumienia tekstu.
- Sprawdź poprawność zdekodowanych danych:

```
% decode

dsig = huffmandeco(comp,dict);
our = huffmandeco(encodedData,dict);

isequal(dsig,our)
```

### 1.3 Związek pomiędzy danymi wejściowymi, ich entropią o współczynniku kompresji.

- Na początku trzeba zdefiniować funkcję obliczającą entropię, która będzie potrafiła przyjąć jako argument wejściowy strumień tekstowy:

```
function H = ComputeEntropy(s)
if (ischar(s)==1) % Checks whether s is a character array
l=length(s);
uniqueChars = unique(s); % String s has all unique characters sorted
lenChar=length(uniqueChars);
```

```

f=zeros(1,lenChar);
for i=1:lenChar
    f(i)=length(findstr(s,uniqueChars(i))); % Count the occurrence of unique
characters
end
p=zeros(1,lenChar);
for i=1:lenChar
    p(i)=f(i)/l; % Probabilities for each unique character
end
H=0;
for i=1:lenChar
    H = H + (-p(i)*log2(p(i))); % Calculating the Entropy
end
else
    display('Invalid String');
end

% function can be found on
https://www.mathworks.com/matlabcentral/fileexchange/38295-compute-the-entropy-of-an-entered-text-string

```

- Zmierz współczynnik entropii dla różnych długości strumieni tekstowych, zapisz je w postaci tabeli, podobnej do tej poniżej:

Lp.	Długość strumienia tekstowego	Entropia	Współczynnik kompresji

- Stwórz następujący strumień tekstowy: "aaaaaaaaabbbbbbbbaaaaaaaaa". Zmierz współczynnik entropii oraz współczynnik kompresji. Zapisz rezultaty.
- Stwórz generowany losowo strumień tekstowy o *długości* = 25 (tak jak tekst z punktu wyżej). Zmierz współczynnik entropii oraz współczynnik kompresji. Zapisz rezultaty.
- Wpisz tekst rzeczywisty (upewnij się, że wszystkie, użyte znaki występują w zdefiniowanym wcześniej alfabecie). Powinien to być rzeczywisty, a nie losowy, tekst! Zmierz współczynnik entropii oraz współczynnik kompresji. Zapisz rezultaty.
- Opisz swoje wnioski.

## **1.4 Ćwiczenia dodatkowe**

Proszę wybrać jedno z poniższych:

- Własna implementacji metody tworzenia słownika Huffmana
- Kompresja bezstratna metoda Huffmana obrazów 2D (przyjmujemy, że są to obrazy w skali szarości).