Informacja i kompresja danych
Wykład 1

Niech $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ będzie niepustym, skończonym zbiorem. Zbiór ten będziemy nazywali **alfabetem**, a jego elementy **literami**.

Słowem w alfabecie $\mathcal{A}$ nazywamy dowolny skończony ciąg elementów zbioru $\mathcal{A}$ (to znaczy sekwencję liter).

Tradycyjnie słowo o literach $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ zapisujemy w postaci

$$a_{i_1} a_{i_2} \ldots a_{i_k},$$

a nie $(a_{i_1}, a_{i_2}, \ldots, a_{i_k})$ (chyba, że ten zapis prowadziłby do niejednoznacznego odczytu).

Dwa słowa $a_{i_1} a_{i_2} \ldots a_{i_k}$ i $b_{j_1} b_{j_2} \ldots b_{j_l}$ w tym samym alfabecie są równe jeśli $k = l$, oraz $a_{i_1} = b_{j_1}, \ldots, a_{i_k} = b_{j_k}$. Na przykład jeśli a i b są różnymi literami, to $ab \neq ba$.

Do zbioru słów dołączamy **słowo puste**, które oznaczamy przez ε.

Długością słowa nazywamy liczbę liter w nim występujących. Słowo puste ma długość 0. Długość słowa v oznaczamy przez $|v|$.

Warto zauważyć, że każdą literę możemy traktować jako słowo długości 1.

Oznaczamy przez $\mathcal{A}^*$ zbiór wszystkich słów w alfabecie $\mathcal{A}$.

Przez $\mathcal{A}^n$ oznaczmy zbiór słów długości n. W szczególności $\mathcal{A}^0 = \{\varepsilon\}$.

Alfabetem binarnym nazywamy zbiór $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, a słowa w tym alfabecie **słowami binarnymi**.

Jeśli słowo składa się z powtarzającej się k razy litery a, to zapisujemy je w postaci a^k, tak więc $a^k = \underbrace{aa \ldots a}_{k\times}$. Tą konwencję stosujemy również bardziej złożonych słowach, których pojawiają się sekwencje powtarzających się takich samych liter stojących obok siebie, na przykład

$$a^2 b^3 = aabbb.$$

Przyjmujemy też konwencję $a^0 = \varepsilon$ dla dowolnej litery a.

Przykłady.

1. Jeśli $\mathcal{A} = \{a\}$, to $\mathcal{A}^* = \{\varepsilon, a, a^2, a^3, a^4, a^5, \ldots\} = \{a^k : \ k \in \mathbb{N} \cup \{0\}\}$.

2. $\Sigma^* = \{0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \ldots\}$.

3. Zbiór liczb naturalnych zapisywanych w systemie dziesiątkowym jest podzbiorem zbioru słów w alfabecie $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

4. Zbór słów w języku polskim jest podzbiorem zbioru słów w alfabecie polskim.

Słowa w danym alfabecie $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ możemy zobrazować przy pomocy drzewa z korzeniem.

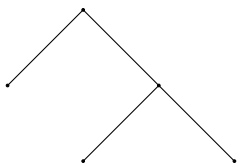
Drzewo z korzeniem jest specjalnym rodzajem grafu nieskierowanego. Tutaj jednak przytoczymy mniej formalną definicję.

Drzewo jak każdy graf składa się z wierzchołków (które będziemy oznaczać jako kropki na płaszczyźnie) oraz krawędzi łączących pary wierzchołków (które będziemy oznaczać jako odcinki). Dodatkowo drzewo jest jest grafem spójnym bez pętli, to znaczy jeśli wybierzemy dwa dowolne różne wierzchołki, to możemy przejść z od pierwszego z nich do drugiego poruszając się po krawędziach oraz nie istnieje połączenie wierzchołka z samym sobą, oprócz połączenia, w którym poruszamy się po krawędzi w jedną i drugą stronę.

Drzewo z korzeniem jest drzewem, w którym wyróżniono jeden wierzchołek (korzeń).

Drzewo z korzeniem rysujemy na płaszczyźnie poziomami od góry. Na poziomie 0 znajduje się korzeń, następnie poniżej na poziomie 1 wszystkie wierzchołki połączone z korzeniem krawędzią, poniżej na poziomie 2 wszystkie, te które połączone są z wierzchołkami poziomu 1 i których jeszcze nie ma na rysunku itd.

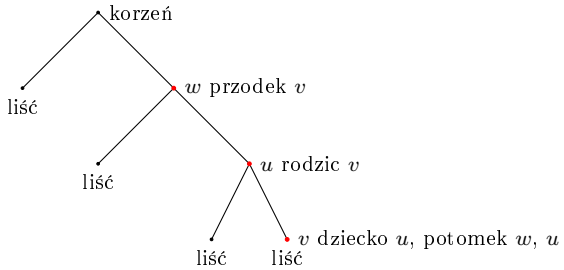
Oto przykład dwupoziomowego drzewa z korzeniem:



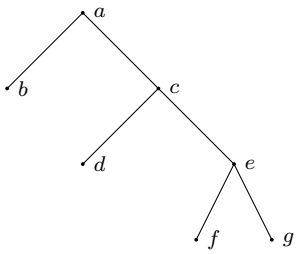
Przodkiem wierzchołka v nazywamy każdy wierzchołek w taki, że można dojść idąc po krawędziach w dół do w do v, a wtedy v nazywamy **potomkiem** w. Jeśli w jest przodkiem v połączonym z nim pojedynczą krawędzią, to w nazywamy **rodzicem lub ojcem** v, a

v **dzieckiem lub synem** w. Wierzchołek, który nie ma potomstwa nazywamy **liściem**.

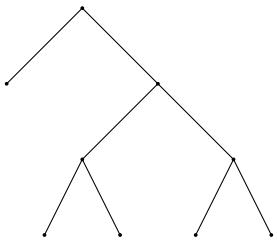
Oczywiście każdy wierzchołek jest potomkiem korzenia.



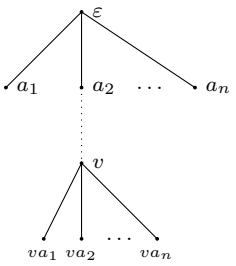
Drzewo nazywamy **etykietowanym** jeśli każdemu wierzchołkowi przyporządkowany jest dodatkowy znacznik (liczba, litera, ciąg znaków).



Drzewo nazywamy **binarnym** jeśli każdy wierzchołek, oprócz liści ma dwójkę dzieci.



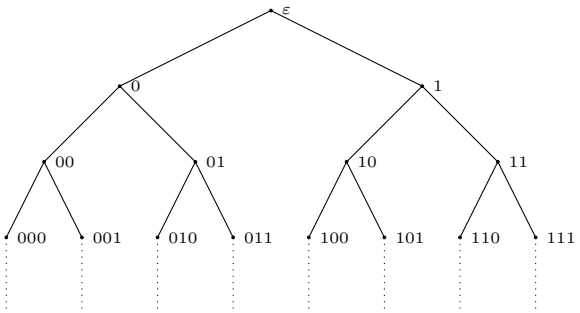
Drzewo słów w alfabecie $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ jest drzewem etykietowanym z korzeniem, w który korzeniem jest wierzchołek z etykietą ε (słowo puste), na poziomie pierwszym umieszczone są litery alfabetu w kolejności $a_1, a_2, \ldots, a_n$, a dziećmi każdego wierzchołka v są $va_1, va_2, \ldots, va_n$ (w tej kolejności).



Etykietę przypisaną wierzchołkowi w tym grafie nazywamy jego **adresem**.

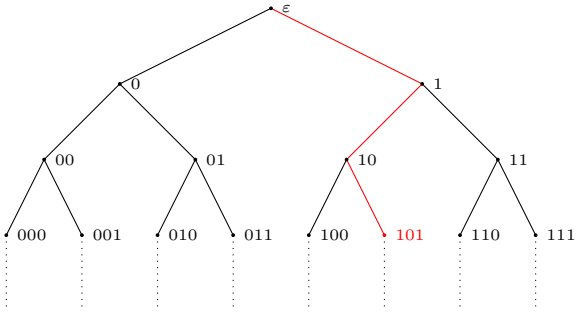
Drzewo, które odpowiada słowom w alfabecie binarnym $\Sigma = \{0, 1\}$ jest **drzewem binarnym**.

Na poniższym rysunku pokazano trzy pierwsze poziomy drzewa słów binarnych.



Nietrudno jest zauważyć, że długość danego słowa jest równa liczbie krawędzi, które prowadzą w dół od korzenia do wierzchołka, którego adresem jest to słowo.

Na przykład długość słowa długość słowa 101 wynosi trzy, bo prowadzą do niego trzy (czerwone) krawędzie od korzenia.



W zbiorze słów w alfabecie $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ wprowadzamy działanie łączenia słów (zwane także dopisywaniem lub konkatenacją). Jeśli $v = a_{i_1} \ldots a_{i_k}$, $w = b_{j_1} \ldots b_{j_l}$, to

$$vw = a_{i_1} \ldots a_{i_k} b_{j_1} \ldots b_{j_l}.$$

Ponadto $v\varepsilon = \varepsilon v = v$ dla każdego $v \in \mathcal{A}^*$.

Na przykład jeśli $v = 1011$, $w = 11101$, to $vw = 1011\,11101$.

Zbiór $\mathcal{A}$ z działaniem konkatenacji jest półgrupą z jedynką (taką półgrupę nazywamy też monoidem), to znaczy działanie konkatenacji jest

łączne:

$$\forall_{v,w,u\in\mathcal{A}^*}\ (vw)u = v(wu)$$

(to jest warunek półgrupy) oraz posiada element neutralny, którym jest ε:

$$\forall_{v\in\mathcal{A}^*}\ v\varepsilon = \varepsilon v = v.$$

Słowo $v \in \mathcal{A}^*$ nazywamy **prefiksem** (lub **przedrostkiem**) słowa $w \in \mathcal{A}^*$ jeśli istnieje $u \in \mathcal{A}^*$ takie, że

$$w = vu.$$

W szczególności słowa ε i w są prefiksami słowa w.

Słowo $v \in \mathcal{A}^*$ nazywamy **sufiksem** (lub **przyrostkiem**) słowa $w \in \mathcal{A}^*$ jeśli istnieje $u \in \mathcal{A}^*$ takie, że

$$w = uv.$$

W szczególności słowa ε i w są sufiksami słowa w.

Na przykład słowo $v = 11011$ jest prefiksem słowa $w = \mathbf{11011}\,110$, a słowo $v = 111$ jest sufiksem słowa $w = 1100\,\mathbf{111}$.

W graficznej reprezentacji słów, prefiksem danego słowa jest każdy jego przodek, a sufiksem każdy jego potomek.

Definicja 1. *Niech $\mathcal{A} = \{a_1, \ldots, a_n\}$ i $\mathcal{B} = \{b_1, \ldots, b_m\}$ będą dwoma alfabetami.* ***Kodem*** *nazywamy dowolną funkcję*

$$f : \mathcal{A} \to \mathcal{B}^*.$$

Wartość $f(a_i)$ każdej litery a_i jest słowem w alfabecie $\mathcal{B}$. Słowa $f(a_1), \ldots, f(a_n)$ nazywamy **słowami kodowymi**.

Czasami kod będziemy utożsamiać, ze zbiorem słów kodowych $\{f(a_1), \ldots, f(a_n)\}$.

Przykład. Niech $\mathcal{A} = \{a, b, c\}$, $\mathcal{B} = \Sigma = \{0, 1\}$. Wtedy funkcja

$$f : \begin{cases} a & \to & 00 \\ b & \to & 110 \\ c & \to & 1011 \end{cases}$$

jest przykładem kodu. Słowami kodowymi sa $00, 110, 1011$.

Funkcja $f : \mathcal{A} \to \mathcal{B}^*$ może być jednoznacznie przedłużona do funkcji $\bar{f} : \mathcal{A}^* \to \mathcal{B}^*$:

$$\bar{f}(a_{i_1}a_{i_2}\dots a_{i_k}) = f(a_{i_1})f(a_{i_2})\dots f(a_{i_k}).$$

Przykład. Niech $\mathcal{A} = \{a, b, c\}$, $\mathcal{B} = \Sigma = \{0, 1\}$. Dany jest kod

$$f : \begin{cases} a & \to & 00 \\ b & \to & 110 \\ c & \to & 1011 \end{cases}$$

Wtedy $\bar{f}(a^2bc) = f(a)f(a)f(b)f(c) = 00\ 00\ 110\ 1011$.

Obliczanie wartości funkcji $\bar{f}(v)$ nazywamy **kodowaniem**, a znajdowanie v jeśli znamy $\bar{f}(v)$ nazywamy **dekodowaniem**.

Definicja 2. *Kod $f : \mathcal{A} \to \mathcal{B}^*$ nazywamy **jednoznacznie dekodowalnym** jeśli funkcja $\bar{f}$ jest różnowartościowa.*

Oznacza to, że mając zakodowany ciąg $w \in \mathcal{B}^*$ znajdziemy dokładnie jedno $v \in \mathcal{A}$, że $\bar{f}(v) = w$.

Oczywiście warunkiem koniecznym dla jednoznacznej dekodowalności jest różnowartościowość funkcji f, ale nie jest warunkiem wystarczającym.

Przykłady.

1. Niech $\mathcal{A} = \{a, b, c, d\}$, $\mathcal{B} = \Sigma = \{0, 1\}$. Kod:

$$f : \begin{cases} a & \to & 00 \\ b & \to & 110 \\ c & \to & 1011 \\ d & \to & 1111. \end{cases}$$

jest jednoznacznie dekodowalny. Dość łatwo można się o tym przekonać. Jeśli podczas dekodowania natrafimy na ciąg odpowiadający słowu kodowemu, to ten ciąg będzie jednoznacznie odczytany, bo żadne ze słów kodowych nie są początkami innych słów kodowych.

Na przykład ciąg 1100010111011110 odczytamy następująco:

$$1100010111011110 \to \underbrace{110}_{b}0010111011110 \to$$

$$\underbrace{110}_{b}\underbrace{00}_{a}10111011110 \to \underbrace{110}_{b}\underbrace{00}_{a}\underbrace{1011}_{c}1011110 \to$$

$$\underbrace{110}_{b}\underbrace{00}_{a}\underbrace{1011}_{c}\underbrace{1011}_{c}110 \to \underbrace{110}_{b}\underbrace{00}_{a}\underbrace{1011}_{c}\underbrace{1011}_{c}\underbrace{110}_{b} \to baccb$$

2. Kod $f : \{a, b, c, d, e\} \to \Sigma^*$:

$$f : \begin{cases} a & \to & 0011 \\ b & \to & 1111 \\ c & \to & 01 \\ d & \to & 00 \\ e & \to & 1101. \end{cases}$$

Nie jest jednoznacznie dekodowalny.

Na przykład ciąg 0011111101 może być odkodowany na dwa sposoby:

$$\begin{aligned} \underbrace{0011}_{a}\underbrace{1111}_{b}\underbrace{01}_{c} &\to abc \\ \underbrace{00}_{d}\underbrace{1111}_{b}\underbrace{1101}_{e} &\to dbe. \end{aligned}$$

Definicja 3. *Jeśli każde słowo kodowe kodu $f : \mathcal{A} \to \mathcal{B}^*$ ma tą samą długość, to kod nazywamy* ***kodem blokowym****.*

Przykłady kodów blokowych.

1. $f : \{a, b, c, d\} \to \Sigma^*$

$$\begin{array}{lcl} a & \to & 00 \\ b & \to & 01 \\ c & \to & 10 \\ d & \to & 11. \end{array}$$

2. Kod ASCII.

Stwierdzenie 1. *Jeśli $f : \mathcal{A} \to \mathcal{B}^*$ jest kodem blokowym i słowa kodowe są parami różne, to f jest jednoznacznie dekodowalny.*

Definicja 4. *Kod $f : \mathcal{A} \to \mathcal{B}^*$ nazywamy* ***kodem prefiksowym*** *(lub* ***kodem przedrostkowym****) jeśli żadne słowo kodowe nie jest przedrostkiem innego słowa kodowego.*

Przykład. $f : \{a, b, c, d\} \to \Sigma^*$

$$\begin{array}{lcl} a & \to & 00 \\ b & \to & 011 \\ c & \to & 101 \\ d & \to & 1111. \end{array}$$

jest kodem prefiksowym.

Stwierdzenie 2. *Każdy kod blokowy o parami różnych słowach kodowych jest kodem prefiksowym.*

Stwierdzenie 3. *Kod prefiksowy jest jednoznacznie dekodowalny.*

Kody prefiksowe nazywamy też **kodami bez opóźnień**. Nazwa bierze się z faktu, że w trakcie dekodowania odczytując zakodowaną treść znak po znaku, jeśli natrafimy na ciąg odpowiadający słowu kodowemu, to możemy tą partię zakodowanego ciągu jednoznacznie odkodować.

Wyjaśnimy, to na dwóch przykładach.

Przykłady.

1. $f : \{a, b, c, d\} \to \Sigma^*$

$$\begin{aligned} a &\to 00 \\ b &\to 011 \\ c &\to 101 \\ d &\to 1111. \end{aligned}$$

Chcemy odkodować ciąg 00000111011011111. Możemy założyć, że kolejne bity przychodzą do nas sukcesywnie.

Po odczytaniu dwóch pierwszych zer 00 wiemy, że informacja zaczęła się od litery a, bo żadne inne słowo kodowe nie zaczyna się od sekwencji 00.

Następnie znów pojawi się sekwencja 00, więc kolejnym znakiem znów będzie a, a potem sekwencję 011 odczytamy jako b.

Ostatecznie po pełnym odkodowaniu otrzymamy $aabccd$.

Zaletą takiego kodu jest fakt, że natrafiając na słowo kodowe nie musimy czekać na pojawienie się kolejnej litery, żeby być przekonanym, że zakodowane zostało właśnie to słowo, a nie inne.

2. $f : \{a, b, c\} \to \Sigma^*$

$$\begin{aligned} a &\to 00 \\ b &\to 1101 \\ c &\to 0011. \end{aligned}$$

Ten kod nie jest prefiksowy (choć jest jednoznacznie dekodowalny).

Na przykład po otrzymaniu wiadomości 0011010011 i po odczytaniu sekwencji 00 nie jesteśmy pewni, czy zakodowany ciąg zaczynał się od a, czy od c. Musimy czekać na nadanie kolejnych znaków. Sprawę pogarsza fakt, że po odczytaniu sekwencji 0011 (która odpowiada literze c) nadal nie jesteśmy pewni, czy zakodowany ciąg zaczynał się od c, czy od sekwencji ab. Dopiero kolejne napływające do nas znaki przekonają nas, że zakodowany został ciąg abc.

A skąd wiemy, że kod w przykładzie 2. jest jednoznacznie dekodowalny?

Definicja 5. *Kod nazywamy* ***kodem sufiksowym*** *jeśli żadne słowo kodowe nie jest końcem innego słowa kodowego.*

Stwierdzenie 4. *Kody sufiksowe są jednoznacznie dekodowalne.*

Ponieważ kod w przykładzie 2. jest sufiksowy, to jest jednoznacznie dekodowalny.

Pocieszające jest to, że WSZYSTKIE kody, które będą miały naturę taką jak w tym wykładzie i które będziemy konstruować na kolejnych wykładach będą prefiksowe. Zatem nie trzeba będzie uzasadniać ich jednoznacznej dekodowalności i dekodowanie będzie prostym procesem.

Mimo powyższej uwagi warto opisać algorytm pozwalający decydować, czy dany kod jest jednoznacznie dekodowalny, czy nie.

Ten algorytm nosi nazwę **algorytm Sardinasa-Petersona**.

Algorytm Sardinasa-Petersona
Jeśli mamy dwa zbiory słów N, D w tym samym alfabecie to określamy zbiór $N^{-1}D$ jako:

$$N^{-1}D = \{y : \ xy \in D, \ x \in N\}.$$

Na przykład jeśli $N = \{00, 11\}$, $D = \{0010, 1101, 0011, 0101\}$, to

$$N^{-1}D = \{10, 01, 11\}.$$

Dany jest kod $f : \mathcal{A} \to \mathcal{B}^*$, o zbiorze słów kodowych C.

Budujemy kolejno zbiory $S_1 = C^{-1}C \setminus \{\varepsilon\}$, $S_{i+1} = S_i^{-1}C \cup C^{-1}S_i$, dla wszystkich $i \geq 1$.

Jeśli $S_i \cap C \neq \emptyset$ lub $\varepsilon \in S_i$ dla pewnego i, to kod nie jest jednoznacznie dekodowalny.

W przeciwnym razie kod jest jednoznacznie dekodowalny.

Warto tu podkreślić, że algorytm ten zawsze zakończy działanie, bo chociaż potencjalnie zbiorów S_i jest nieskończenie wiele, to nastąpi taki moment, że te zbiory zaczną sie powtarzać.

Warto też podkreślić, że algorytm działa w czasie wielomianowym.

Przykłady.

1. Rozważmy kod $f : \{a, b, c, d, e\} \to \Sigma^*$:

$$f : \begin{cases} a & \to & 1100 \\ b & \to & 0000 \\ c & \to & 10 \\ d & \to & 11 \\ e & \to & 0001. \end{cases}$$

Zatem $C = \{1100, 0000, 10, 11, 0001\}$.
Budujemy zbiory S_i i sprawdzamy warunki $S_i \cap C \neq \emptyset$ lub $\varepsilon \in S_i$:

- $S_1 = C^{-1}C \setminus \{\varepsilon\} = \{00\}$, $S_1 \cap C = \emptyset$.
- $S_2 = S_1^{-1}C \cup C^{-1}S_1 = \{00, 01\}$, $S_1 \cap C = \emptyset$.
- $S_3 = S_2^{-1}C \cup C^{-1}S_2 = \{00, 01\}$, $S_2 \cap C = \emptyset$.
- Algorytm kończy działanie, bo ponieważ $S_3 = S_2$, to $S_i = S_2$ dla wszystkich $i \geq 2$. Zatem $S_i \cap C = \emptyset$ dla wszystkich i.

To oznacza, że f jest jednoznacznie dekodowalny.

2. Rozważmy kod $f : \{a, b, c, d, e\} \to \Sigma^*$:

$$f : \begin{cases} a & \to & 1100 \\ b & \to & 0000 \\ c & \to & 10 \\ d & \to & 11 \\ e & \to & 0010. \end{cases}$$

Zatem $C = \{1100, 0000, 10, 11, 0010\}$.
Budujemy zbiory S_i i sprawdzamy warunki $S_i \cap C \neq \emptyset$ lub $\varepsilon \in S_i$:

- $S_1 = C^{-1}C \setminus \{\varepsilon\} = \{00\}$, $S_1 \cap C = \emptyset$.
- $S_2 = S_1^{-1}C \cup C^{-1}S_1 = \{00, 10\}$, $S_1 \cap C = \{10\}$.
- Zatem kod nie jest jednoznacznie dekodowalny.

Jeśli kod jest prefiksowy, to $C^{-1}C = \emptyset$, więc w pierwszym kroku Algorytmu Sardinasa-Petersona otrzymujemy wynik.

Dane są alfabety $\mathcal{A} = \{a_1, \ldots, a_n\}$ i $\mathcal{B} = \{b_1, \ldots, b_m\}$ oraz liczby naturalne $l_1, \ldots, l_n$. Pytanie brzmi: Jaki warunek musza spełniać liczby $l_1, \ldots, l_n$ aby istniał kod jednoznacznie dekodowalny (lub kod bez opóźnień) $f : \mathcal{A} \to \mathcal{B}^*$ o długościach słów kodowych $l_1, \ldots, l_n$?

Jeśli założymy, że kod jest blokowy, to $l_1 = l_2 = \ldots = l_n = l$. Liczba słów długości l w alfabecie $\mathcal{B}$ wynosi m^l. Jeśli kod ma być jednoznacznie dekodowalny, to liczba liter nie może przewyższać liczby m^l. Zatem istnieje jednoznacznie dekodowalny kod blokowy o długościach słów

kodowych równych l wtedy i tylko wtedy, gdy

$$k \leq m^l.$$

Tą nierówność możemy zapisać w innej formie:

$$\frac{k}{m^l} \leq 1,$$

a używając równości $l_1 = l_2 = \ldots = l_n = l$ możemy ją zapisać w postaci:

$$\sum_{i=1}^{n} \frac{1}{m^{l_i}} \leq 1.$$

Tą ostatnią nierówność nazywamy **nierównością Krafta** i jest ona warunkiem koniecznym i dostatecznym na istnienie kodu jednoznacznie dekodowalnego (i również kodu bez opóźnień.

Twierdzenie 1 (Kraft, McMillan)**.** *Dane są alfabety* $\mathcal{A} = \{a_1, \ldots, a_n\}$ *i* $\mathcal{B} = \{b_1, \ldots, b_m\}$ *oraz liczby naturalne* $l_1, \ldots, l_n$*. Wtedy następujące warunki są równoważne:*

1. $\sum_{i=1}^{n} \frac{1}{m^{l_i}} \leq 1$.

2. *Istnieje kod bez opóźnień* $f : \mathcal{A} \to \mathcal{B}^*$ *o długościach słów kodowych* $l_1, \ldots, l_n$.

3. *Istnieje kod jednoznacznie dekodowalny* $f : \mathcal{A} \to \mathcal{B}^*$ *o długościach słów kodowych* $l_1, \ldots, l_n$.

Przykład. Niech $\mathcal{A} = \{a, b, c, d\}$, $\mathcal{B} = \Sigma$. Czy istnieje kod bez opóźnień $f : \mathcal{A} \to \mathcal{B}^*$ o długościach słów kodowych $1, 2, 3, 3$?

Tak, taki kod istnieje, bo:

$$\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^3} = 1.$$

Spełniony jest więc warunek Krafta.

Przykładem takiego kodu jest

$$f : \begin{cases} a & \to & 1 \\ b & \to & 00 \\ c & \to & 010 \\ d & \to & 011. \end{cases}$$

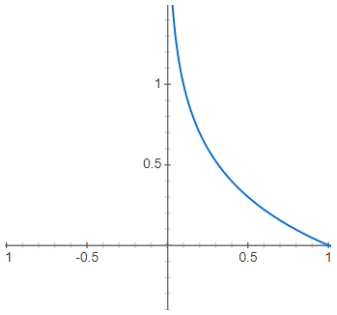
Informacja i kompresja danych
Wykład 2

Definicja 1. Ilością informacji (lub autoinformacją) zdarzenia x zachodzącego z prawdopodobieństwem $p \in [0, 1]$ nazywamy liczbę:

$$I(p) = -\log p.$$

Przyjmujemy, że podstawą logarytmu $\log x$ jest 2.

Funkcja $I(p) = -\log p$ ma wykres:



To oznacza, że im większe prawdopodobieństwo zajścia zdarzenia, tym mniejszą wartość informacyjną niesie ze sobą pojawienie się tego zdarzenia. To jest naturalne, bo stwierdzenie, że tego dnia wzeszło słońce ma dla nas mniejszą wartość, niż informacja o wygranej w grze liczbowej.

Jeśli logarytm ma podstawę 2, to jednostką ilości informacji jest bit, przy podstawie e nat, a przy podstawie 10 hartley.

Definicja 2. Źródłem nazywamy alfabet $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ wraz z przypisanymi poszczególnym literom prawdopodobieństwami $p_1, \ldots, p_n$ takimi, że

$$\sum_{i=1}^{n} p_i = 1.$$

Jeśli $P = \{p_1, \ldots, p_n\}$, to źródło będziemy zapisywali jako parę uporządkowaną $(\mathcal{A}, P)$.

Prawdopodobieństwo p_i traktujemy jako prawdopodobieństwo pojawienia się znaku a_i (lub zajścia zdarzenia a_i).

Źródło można interpretować jako opis wszystkich możliwych zdarzeń związanych z pewnym zjawiskiem.

Przykłady.

1. Przypuśćmy, że źródło wysyła dane o stanie pogody w pewnej krainie. W tej krainie są trzy stany pogodowe: jest słonecznie, pada deszcz, jest duże zachmurzenie, ale nie pada. Każdemu z tych stanów pogodowych można przypisać prawdopodobieństwo jego wystąpienia:
- jest słonecznie - $\frac{1}{12}$.
- pada deszcz - $\frac{1}{2}$.
- zachmurzenie - $\frac{5}{12}$.

2. Źródło emituje wyniki meczów Polska-Niemcy. Z dotychczasowych spotkań wynikają następujące prawdopodobieństwa:
- Polska wygra - $\frac{1}{21}$.
- będzie remis - $\frac{7}{21}$.
- Polska przegra - $\frac{13}{21}$.

Definicja 3. Źródło $(\mathcal{A}, \{p_1, \ldots, p_n\})$ nazywamy **źródłem bez pamięci** jeśli przy nadawaniu sekwencji znaków z alfabetu $\mathcal{A}$ kolejne znaki pojawiają się niezależnie od znaków pojawiających się wcześniej.

To znaczy litery ze zbioru $\mathcal{A}$ są niezależnymi zdarzeniami.

Definicja 4. Jeśli $(\mathcal{A}, \{p_1, \ldots, p_n\})$ jest źródłem bez pamięci, to w zbiorze $\mathcal{A}^n$ możemy zdefiniować prawdopodobieństwa:

$$p(a_{i_1} \ldots a_{i_n}) = p_{i_1} \cdots p_{i_n}.$$

Zbiór tych prawdopodobieństw oznaczamy przez P^n.
W ten sposób otrzymujemy źródło $(\mathcal{A}^n, P^n)$.

Sprawdzimy na przykładzie P^2, że suma prawdopodobieństw w tym zbiorze jest równa 1:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} p_i p_j = \sum_{i=1}^{n} p_i \underbrace{\left(\sum_{j=1}^{n} p_j\right)}_{=1} = \sum_{i=1}^{n} p_i = 1.$$

Definicja 5. Entropią (lub średnią ilością informacji) źródła $(\{a_1, \ldots, a_n\}, \{p_1, \ldots, p_n\})$ nazywamy liczbę:

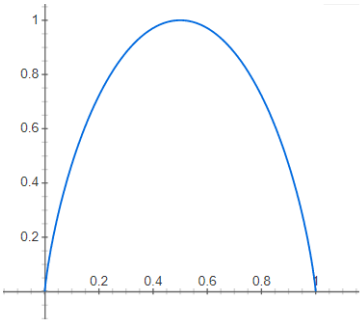
$$H(\mathcal{A}) = -\sum_{i=1}^{n} p_i \log p_i = \sum_{i=1}^{n} p_i \log \frac{1}{p_i}.$$

Entropię źródła nazywamy też **średnią niepewnością odbiorcy oczekującego na wyjście ze źródła**.

Przykład. Przypuśćmy, że źródło nadaje dwie wiadomości a_1, a_2 o prawdopodobieństwach p, q. Ponieważ $p+q = 1$, to $q = 1-p$ i entropia wynosi:

$$H(p) = -p\log p - (1-p)\log (1-p).$$

Wykres tej funkcji jest następujący:



Zauważmy, że ta funkcja przyjmuje najwyższą wartość 1 dla $p = q = \frac{1}{2}$ i jest tym większa im bliższe są wartości p, q. Natomiast dla $p = 0, q = 1$ i $p = 1, q = 0$ entropia wynosi 0.

Jest to naturalne, bo im bliższe są wartości p, q tym większą niepewność ma obserwator oczekujący na wyjście ze źródła i tym ciekawsze są pojawiające się wyniki.

Przykład. Które źródło uznamy za ciekawsze: nadające wyniki meczów Niemcy-Gibraltar, czy nadające wyniki meczów Niemcy-Brazylia?

Stwierdzenie 1. *Jeśli źródło $(\mathcal{A}, P)$ jest bez pamięci, to*

$$H(\mathcal{A}^n) = nH(\mathcal{A}).$$

Możemy też zdefiniować entropię przy pomocy logarytmów przy podstawie m:

$$H_m(\mathcal{A}) = -\sum_{i=1}^{n} p_i \log_m p_i.$$

Ze wzoru $\log_m p = \frac{\log p}{\log m}$ wynika, że

$$H_m(\mathcal{A}) = \frac{H(\mathcal{A})}{\log m}.$$

Własności entropii

$$0 \leq H(\mathcal{A}) \leq \log n.$$

Dodatkowo $H(\mathcal{A}) = 0$ tylko wtedy, gdy jedna z liczb p_i jest równa jeden, a pozostałe 0, a $H(\mathcal{A}) \leq \log n$ tylko wtedy, gdy $p_1 = \ldots = p_n = \frac{1}{n}$.

Lemat 1. *Dla każdego* $x \in (0, \infty)$*:*

$$\ln x \leq x - 1.$$

Definicja 6. Dane są źródło: $(\mathcal{A}, P)$, gdzie $\mathcal{A} = \{a_1, \ldots, a_n\}$, P składa się z prawdopodobieństw $p_1, \ldots, p_n$, alfabet $\mathcal{B}$ oraz jednoznacznie dekodowalny kod $f : \mathcal{A} \to \mathcal{B}^*$ o długościach słów kodowych $l_1, \ldots, l_n$. Wtedy liczbę

$$L(f) = \sum_{i=1}^{n} p_i l_i$$

nazywamy **średnią długością słów kodowych**.

Definicja 7. Kod $f : \mathcal{A} \to \mathcal{B}^*$ nazywamy **zwięzłym** (albo **optymalnym**) jeśli dla każdego jednoznacznie dekodowalnego kodu $f' : \mathcal{A} \to \mathcal{B}^*$ zachodzi nierówność

$$L(f) \leq L(f').$$

Lemat 2. *Jeśli* $p_1, \ldots, p_n$ *oraz* $q_1, \ldots, q_n$ *sa nieujemnymi liczbami rzeczywistymi takimi, że* $\sum_{i=1}^{n} p_i = \sum_{i=1}^{n} q_i = 1$*, to*

$$\sum_{i=1}^{n} p_i \log \frac{1}{p_i} \leq \sum_{i=1}^{n} p_i \log \frac{1}{q_i}.$$

Twierdzenie 1. *Dla każdego jednoznacznie dekodowalnego kodu* $f : \mathcal{A} \to \mathcal{B}^*$ *określonego na źródle* $(\mathcal{A}, P)$ *spełniona jest nierówność*

$$\frac{H(\mathcal{A})}{\log m} \leq L(f),$$

gdzie $m = |\mathcal{B}|$*.*

Twierdzenie 2. *Jeśli* $f : \mathcal{A} \to \mathcal{B}^*$ *jest kodem zwięzłym określonym na źródle* $(\mathcal{A}, P)$*, to spełniona jest nierówność*

$$\frac{H(\mathcal{A})}{\log m} \leq L(f) \leq \frac{H(\mathcal{A})}{\log m} + 1.$$

gdzie $m = |\mathcal{B}|$*.*

Niech $(\mathcal{A}, P)$ będzie źródłem bez pamięci i niech $f_n : \mathcal{A}^n \to \mathcal{B}^*$ będzie kodem optymalnym. Wtedy stosując powyższe nierówności otrzymujemy:

$$\frac{H(\mathcal{A}^n)}{\log m} \leq L(f_n) \leq \frac{H(\mathcal{A}^n)}{\log m} + 1.$$

Stąd, wykorzystując fakt, że $H(\mathcal{A}^n) = nH(\mathcal{A})$:

$$n\frac{H(\mathcal{A})}{\log m} \leq L(f_n) \leq n\frac{H(\mathcal{A})}{\log m} + 1.$$

Dzielimy przez n i otrzymujemy:

$$\frac{H(\mathcal{A})}{\log m} \leq \frac{L(f_n)}{n} \leq \frac{H(\mathcal{A})}{\log m} + \frac{1}{n}.$$

Obliczając granicę i korzystając z twierdzenia o trzech ciągach dostajemy:

$$\lim_{n\to\infty} \frac{L(f_n)}{n} = \frac{H(\mathcal{A})}{\log m}.$$

Twierdzenie 3 (Pierwsze Twierdzenie Shannona)**.**

$$\lim_{n\to\infty} \frac{L(f_n)}{n} = \frac{H(\mathcal{A})}{\log m}.$$

Definicja 8. Efektywnością kodu f nazywamy liczbę

$$\mathrm{Ef}(f) = \frac{H(\mathcal{A})}{L(f)} \cdot 100\%.$$

Jeśli x jest liczbą rzeczywistą, to $\lceil x \rceil$ jest najmniejszą liczbą całkowitą wśród liczb większych bądź równych od x. Na przykład $\lceil \pi \rceil = 4$.

Jeśli x jest liczbą rzeczywistą, to $\lfloor x \rfloor$ jest największą liczbą całkowitą wśród liczb mniejszych bądź równych od x. Na przykład $\lfloor \pi \rfloor = 3$.

Każdą liczbę całkowitą z można jednoznacznie zapisać w postaci:

$$z = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_1 2 + a_0,$$

gdzie $a_n, a_{n-1}, \ldots, a_0 \in \{0, 1\}$.

Algorytm

Chcemy zapisać liczbę m w systemie pozycyjnym przy podstawie m.

(1) Przyjmujemy $m_0 = m$ i wyznaczamy resztę z dzielenia m_0 przez n. Tą resztę oznaczamy przez a_0.

(2) Przyjmujemy $m_i = \frac{m_{i-1} - a_{i-1}}{n}$ dla $i > 0$. Obliczamy resztę a_i z dzielenia m_i przez n.

(3) Kroki z poprzedniego punktu wykonujemy tak długo aż natrafimy na k takie, że $m_k < n$ (wtedy $m_{k+1} = 0$). Obliczamy ostatnią resztę a_k.

(4) Otrzymane przedstawienie, to $m = (a_k \ldots a_0)_n$.

Zwykle kroki tego algorytmu przedstawiamy przy pomocy tabelki:

$$\begin{array}{l|l} m_0 = m & a_0 \\ m_1 = \frac{m_0 - a_0}{n} & a_1 \\ m_2 = \frac{m_1 - a_1}{n} & a_2 \\ \vdots & \vdots \\ m_i = \frac{m_{i-1} - a_{i-1}}{n} & a_i \\ \vdots & \vdots \\ m_k = \frac{m_{k-1} - a_{k-1}}{n} & a_k \\ 0 & \end{array}$$

$m = (a_k \dots a_0)_n$.

Przykład. Zapisać liczbę 347 w systemach przy podstawie 3 i 2.

Przy podstawie 3:

$$\begin{array}{r|l} 347 & 2 \\ 115 & 1 \\ 38 & 2 \\ 12 & 0 \\ 4 & 1 \\ 1 & 1 \\ 0 & \end{array}$$

Zatem $347 = (110212)_3 = 3^5 + 3^4 + 2 \cdot 3^2 + 3 + 2$.

Przy podstawie 2:

347	1
173	1
86	0
43	1
21	1
10	0
5	1
2	0
1	1
0	

Zatem $347 = (101011011)_2 = 2^8 + 2^6 + 2^4 + 2^3 + 2 + 1$.

Liczbę $p \in (0,1)$ możemy zapisać jednoznacznie w postaci: $\frac{b_1}{2} + \frac{b_2}{2^2} + \frac{b_3}{2^3} + \ldots$, gdzie $b_1, b_2, \ldots \in \{0,1\}$ i to nazywamy jej rozwinięciem dwójkowym.

Algorytm. Dana jest liczba $p \in [0,1]$ chcemy ją zapisać w postaci $\frac{b_1}{2} + \frac{b_2}{2^2} + \frac{b_3}{2^3} + \ldots$. W tym celu wykonujemy kroki:
(1) Przyjmujemy $s_1 = p$ i wstawiamy $i = 1$,
(2) Obliczamy $t_i = 2 \cdot s_i$,
(3) Przyjmujemy $b_i = \lfloor t_i \rfloor$,
(4) $s_{i+1} = t_i - b_i$, $i = i + 1$,
(5) jeśli $s_i \neq 0$ wracamy do punktu 2, a jeśli $s_i = 0$ kończymy.

Przykład. Zapisać liczbę $\frac{1}{5}$ w systemie binarnym.

- $s_1 = \frac{1}{5}$, $t_1 = 2 \cdot s_1 = \frac{2}{5}$, $b_1 = 0$,
- $s_2 = t_1 - b_1 = \frac{2}{5}$, $t_2 = 2 \cdot t_1 = \frac{4}{5}$, $b_2 = 0$,
- $s_3 = t_2 - b_2 = \frac{4}{5}$, $t_3 = 2 \cdot t_2 = \frac{8}{5}$, $b_3 = 1$,
- $s_4 = t_3 - b_3 = \frac{3}{5}$, $t_4 = 2 \cdot t_3 = \frac{6}{5}$, $b_4 = 1$,
- ponieważ $s_5 = t_4 - b_4 = \frac{1}{5} = s_1$, to wyniki zaczną się powtarzać,
- $\frac{1}{5} = (0, (0011))_2$.

Kod Shannona

Dane jest źródło $(\{a_1, \ldots, a_n\}, \{p_1, \ldots, p_n\})$. Kod, który tworzymy jest binarny.

- Prawdopodobieństwa ustawiamy w ciąg nierosnący $p_1 \geq p_2 \geq p_3 \geq \ldots \geq p_n$.
- Tworzymy prawdopodobieństwa kumulatywne $P_0 = 0$, $P_i = p_1 + \ldots + p_{i-1}$ dla $i \geq 2$.
- Obliczamy $l_i = \lceil -\log p_i \rceil$.
- Literę a_i kodujemy na l_i rozwinięcia binarnego liczby P_i.

Przykład. Alfabet składa się z liter a, b, c, d, e, f, z którymi stowarzyszone są prawdopodobieństwa $0,35$, $0,17$, $0,17$, $0,16$, $0,15$.

a_i	p_i	P_i	l_i	Rozwinięcie	Kod
a	$0,35$	0	2	$0,000000\ldots$	00
b	$0,17$	$0,35$	3	$0,010110\ldots$	010
c	$0,17$	$0,52$	3	$0,100000\ldots$	100
d	$0,16$	$0,69$	3	$0,101100\ldots$	101
e	$0,15$	$0,85$	3	$0,110110\ldots$	110

$H(\mathcal{A}) = -(0,35\log 0,35 + 0,17\log 0,17 + 0,16\log 0,17 + 0,16\log 0,16 + 0,15\log 0,15)$
$\approx 2,2324$,
$L(f) = 0,35 \cdot 2 + 0,17 \cdot 3 + 0,17 \cdot 3 + 0,16 \cdot 3 + 0,15 \cdot 3 = 2,65$,
$\mathrm{Ef}(f) = 84,26\%$.

Kod Shannona-Fano

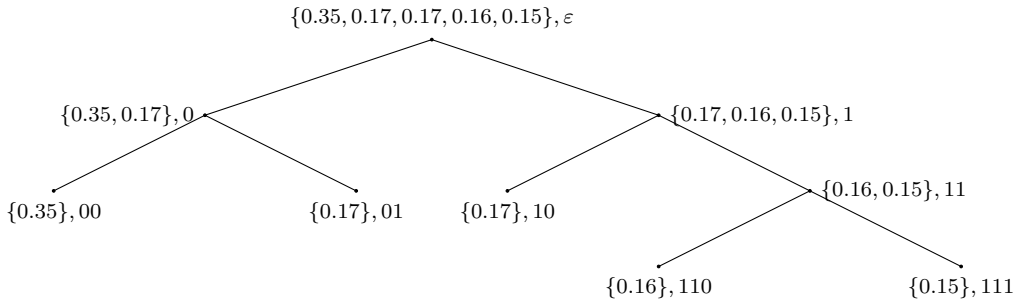
Dane jest źródło $(\{a_1, \ldots, a_n\}, \{p_1, \ldots, p_n\})$. Kod, który tworzymy jest binarny.

- Prawdopodobieństwa ustawiamy w ciąg nierosnący $p_1 \geq p_2 \geq p_3 \geq \ldots \geq p_n$.
- Zbiór $P_0 = \{p_1, \ldots, p_n\}$ dzielimy na dwa podzbiory $\{p_1, \ldots, p_i\}$, $\{p_{i+1}, \ldots, p_n\}$ tak, aby różnica

$$\left| \sum_{s=1}^{i} p_s - \sum_{s=i+1}^{n} p_s \right|$$

  była najmniejsza.
- Następnie każdy ze zbiorów $\{p_1, \ldots, p_i\}$ i $\{p_{i+1}, \ldots, p_n\}$ dzielimy na dwa według tej samej zasady (jeśli te zbiory nie są jednoelementowe).
- Podziały kontynuujemy tak długo aż wszystkie zbiory będą jednoelementowe.
- Zbiorowi $P_0 = \{p_1, \ldots, p_n\}$ przypisujemy kod ε.
- Jeśli dany zbiór P ma kod v i dzieli się na podzbiory P_1, P_2, to zbiór P_1 będzie miał kod $v0$, a P_2 kod $v1$.
- Ostatecznym kodem litery x jest kod przypisany zbiorowi $\{x\}$.

Przykład. Alfabet składa się z liter a, b, c, d, e, f, z którymi stowarzyszone są prawdopodobieństwa $0,35$, $0,17$, $0,17$, $0,16$, $0,15$. Kolejne podziały i odpowiadające im kody będziemy opisywać w postaci drzewa binarnego.



Otrzymaliśmy, więc kod

$$\begin{array}{lcl} a & \to & 00 \\ b & \to & 01 \\ c & \to & 10 \\ d & \to & 110 \\ e & \to & 111. \end{array}$$

$H(\mathcal{A}) = -(0,35\log\, 0,35 + 0,17\log\, 0,17 + 0,16\log\, 0,17 + 0,16\log\, 0,16 + 0,15\log\, 0,15)$
$\approx 2,2324,$
$L(f) = 0,35 \cdot 2 + 0,17 \cdot 2 + 0,17 \cdot 2 + 0,16 \cdot 3 + 0,15 \cdot 3 = 2,31,$
$\mathrm{Ef}(f) = 96,66\%.$

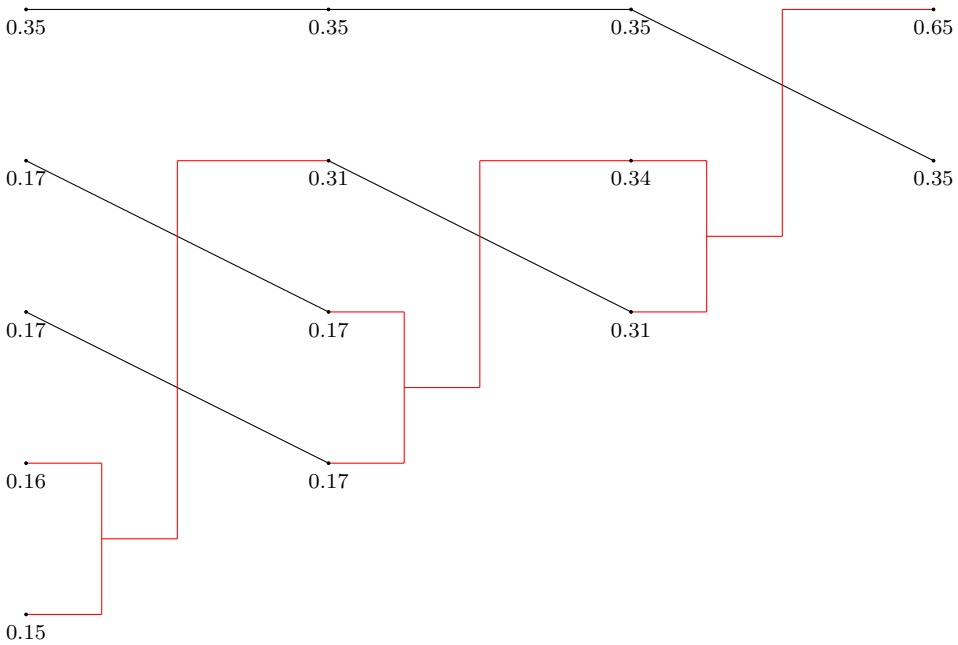
Kod Huffmana

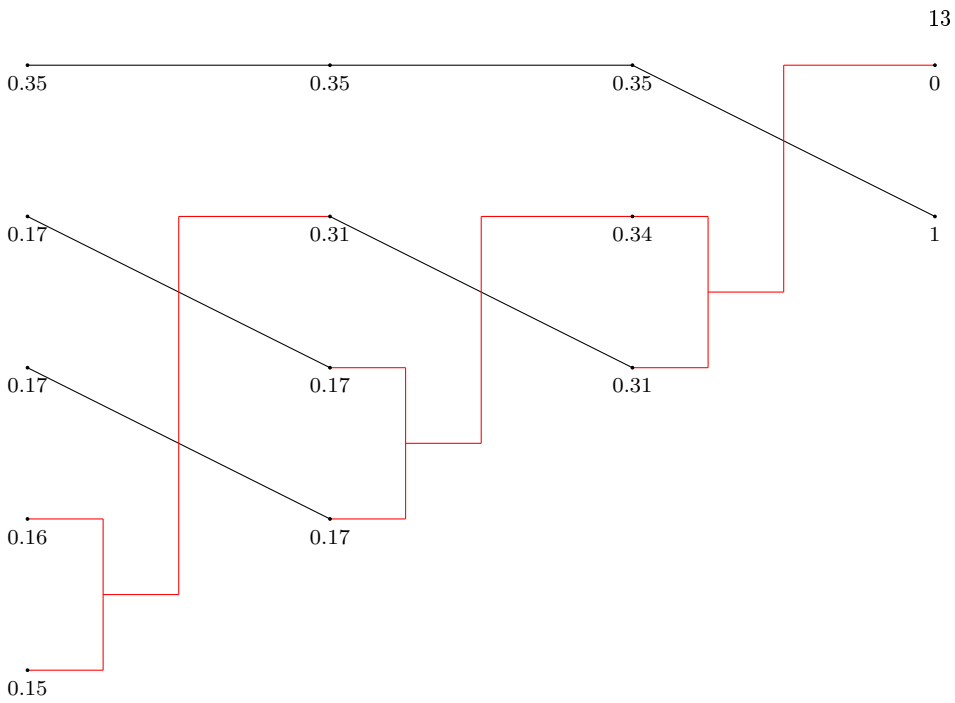
Dane jest źródło $(\{a_1, \ldots, a_n\}, \{p_1, \ldots, p_n\})$. Kod, który tworzymy jest binarny.

- Prawdopodobieństwa ustawiamy w ciąg nierosnący $p_1 \geq p_2 \geq p_3 \geq \ldots \geq p_n$.
- Źródło oznaczamy przez $S_0 = (\{a_1^{(0)}, \ldots, a_n^{(0)}\}, \{p_1^{(0)}, \ldots, p_n^{(0)}\})$. Przyjmujemy $i = 0$.
- Ze źródła $S_i = (\{a_1^{(i)}, \ldots, a_{n-i}^{(i)}\}, \{p_1^{(i)}, \ldots, p_{n-i}^{(i)}\})$ tworzymy źródło $S_{i+1} = (\{a_1^{(i+1)}, \ldots, a_{n-i-1}^{(i+1)}\}, \{p_1^{(i+1)}, \ldots, p_{n-i-1}^{(i+1)}\})$ łącząc dwie najmniej prawdopodobne litery, a ich prawdopodobieństwa dodając do siebie i porządkując je nierosnąco.
- Po $n-2$ krokach otrzymamy źródło dwuliterowe
$$S_{n-1} = (\{a_1^{(n-2)}, a_2^{(n-2)}\}, \{p_1^{(n-2)}, p_2^{(n-2)}\}),$$
w którym $p_1^{(n-2)} \geq p_2^{(n-2)}$.
- Kod tworzymy zaczynając od źródła S_{n-1}. Literę $a_1^{(n-2)}$ kodujemy na 0, a $a_2^{(n-2)}$ kodujemy na 1. Następnie tworzymy kod kolejnych źródeł cofając się od S_{n-1} do S_0.
- Jeśli w trakcie przejścia od S_i do S_{i+1} litera a przeszła na literę b, to kody tych liter są identyczne. Jeśli litery a, b skleiły się w literę c i c ma kod v, to a ma kod $v0$, a b ma kod $v1$.

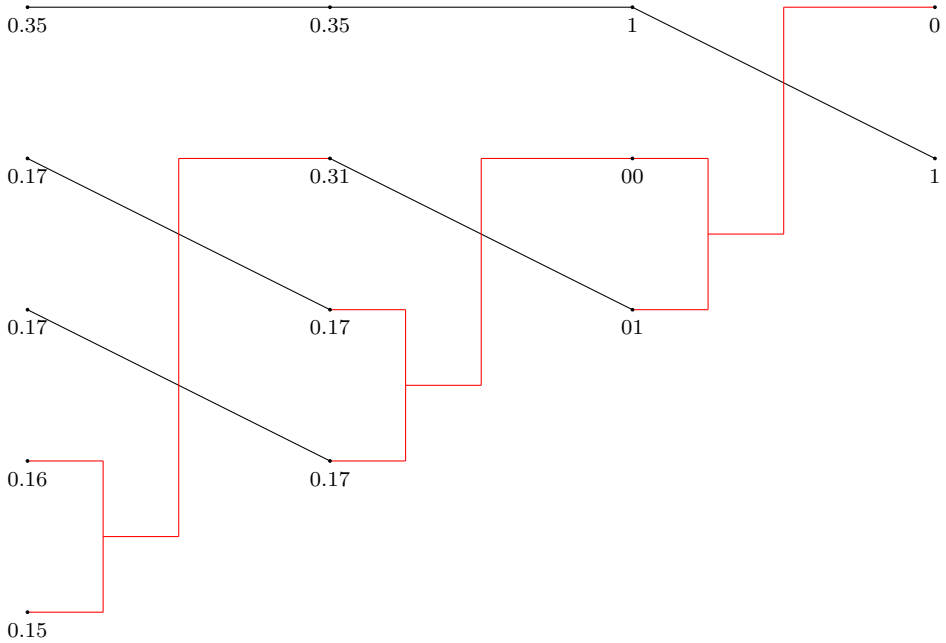
Przykład. Alfabet składa się z liter a, b, c, d, e, f, z którymi stowarzyszone są prawdopodobieństwa $0,35$, $0,17$, $0,17$, $0,16$, $0,15$.

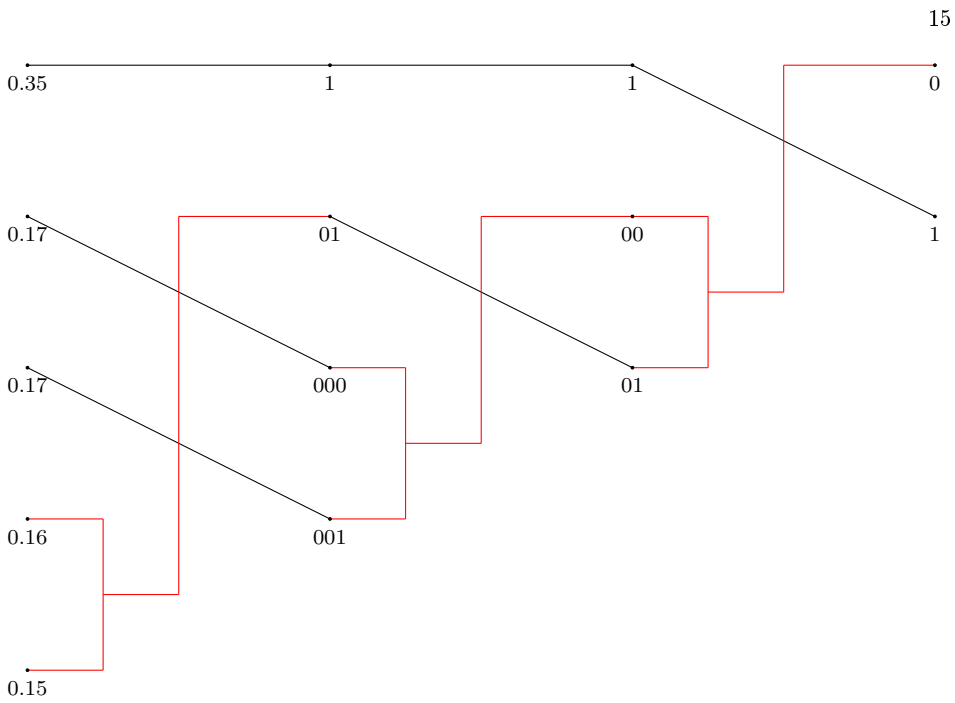
12

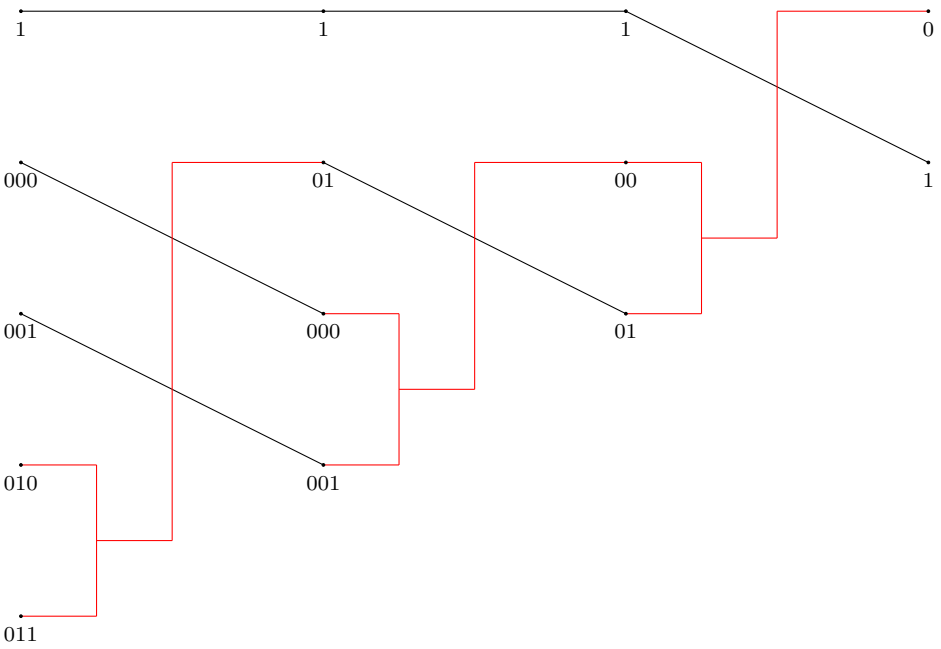



13
0.35
0.35
0.35
0
0.17
0.31
0.34
1
0.17
0.17
0.31
0.16
0.17
0.15

14




15
0.35
1
1
0
0.17
01
00
1
0.17
000
01
0.16
001
0.15



Otrzymaliśmy, więc kod

$$\begin{array}{rcl} a & \to & 1 \\ b & \to & 000 \\ c & \to & 001 \\ d & \to & 010 \\ e & \to & 011. \end{array}$$

$H(\mathcal{A}) = -(0,35\log 0,35 + 0,17\log 0,17 + 0,16\log 0,17 + 0,16\log 0,16 + 0,15\log 0,15)$
$\approx 2,2324,$
$L(f) = 0,35 \cdot 1 + 0,17 \cdot 3 + 0,17 \cdot 3 + 0,16 \cdot 3 + 0,15 \cdot 3 = 2,3,$
$\mathrm{Ef}(f) = 97,08\%.$

Twierdzenie 4. *Kod Huffmana jest optymalny.*

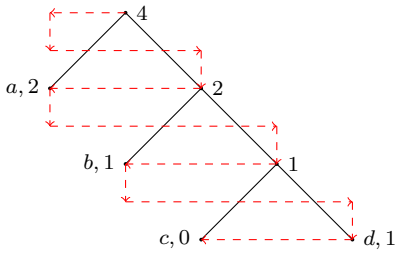
Informacja i kompresja danych
Wykład 3

Dynamiczny kod Huffmana

W dynamicznym kodzie Huffmana przeglądamy otrzymany tekst kodując go (binarnie) znak po znaku. Długość kodu kolejnego znaku jest zmienna i zależy od liczby wystąpień tego znaku w dotychczas odczytanej wiadomości.

Definicja 1. Etykietowane drzewo z korzeniem nazywamy **drzewem z własnością rodzeństwa** jeśli każdemu wierzchołkowi, przypisany jest licznik taki, że każdy wierzchołek, który nie jest liściem ma dwóch potomków, których liczniki sumują się do licznika tego wierzchołka i przechodzenie drzewa wzdłuż od strony prawej do lewej daje nierosnący ciąg liczników.
Dodatkowo każdemu liściowi przyporządkowana jest litera, należąca do pewnego alfabetu.

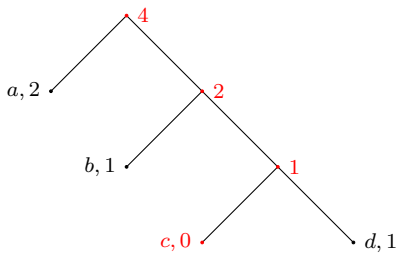
Przykład.



Wierzchołki drzewa z własnością rodzeństwa możemy podzielić na bloki. Blok $blok_i$ składa się ze wszystkich wierzchołków, którym przypisany jest licznik i. Na przykład powyższe drzewo składa się z czterech bloków $blok_0$, $blok_1$, $blok_2$ i $blok_4$.

Liderem bloku $blok_i$ nazywamy ten z wierzchołków tego bloku, który jest położony najwyżej i najbardziej na prawo.

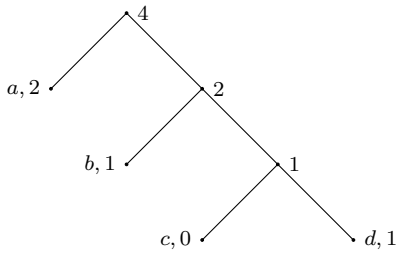
Przykład. Na poniższym rysunku na czerwono zaznaczono liderów poszczególnych bloków.



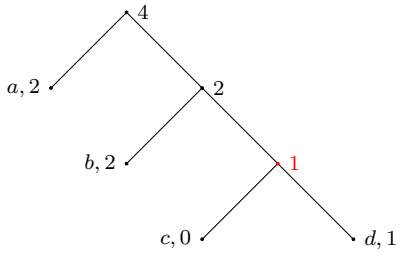
Każdej literze na powyższym drzewie przypisujemy kod odpowiadający adresowi liścia oznaczonego tą literą. Na przykład kodem litery c jest 110.

W trakcie kodowania pobieramy kolejne litery i umieszczamy je na drzewie przypisując im liczniki dotychczasowych wystąpień. Jeśli litera była już pobrana i ma przypisany licznik i (czyli należy do bloku $blok_i$), to zwiększamy jej licznik o jeden. To jednak może zaburzyć własność rodzeństwa. W takim razie należy drzewo zmodyfikować. Modyfikacja polega na zamianie miejscami węzła pobranej litery z liderem bloku $blok_i$, pod warunkiem, że ten lider nie jest rodzicem węzła tej litery. Po dokonaniu przekształceń zwiększamy o jeden liczniki wszystkich przodków węzła pobranej litery.

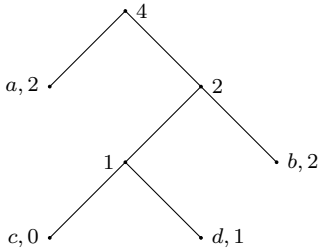
Przykład. Przypuśćmy, że przed pobraniem litery drzewo liczników wystąpień jest następujące:



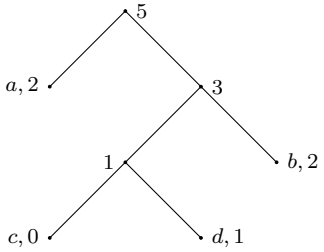
Jeśli kolejnym pobranym znakiem jest b, to w nowym drzewie licznik b zwiększy się o 1:



To jednak spowoduje utratę własności rodzeństwa. Musimy go zamienić z liderem bloku $blok_1$, który jest zaznaczony na czerwono:



Teraz musimy liczniki przodków węzła b zwiększyć o jeden

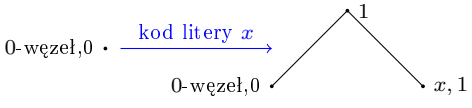


Zauważmy, że licznik korzenia jest równy ilości pobranych dotychczas znaków.

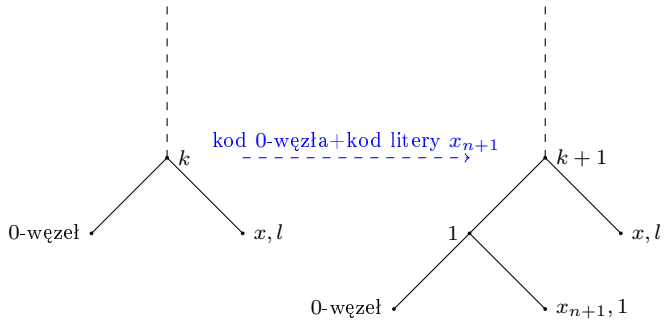
Algorytm

Kodujemy ciąg znaków należących do alfabetu $\mathcal{A}$. Kod tworzymy przechodząc ten ciąg od lewej strony do prawej. W każdym kroku uzupełniamy kod oraz tworzymy drzewo z własnością rodzeństwa, którego wierzchołki (tutaj zwane węzłami) odpowiadają znakom należącym do danego alfabetu. Każdemu wierzchołkowi przypisana jest litera oraz licznik pojawień się tej litery w dotychczasowym tekście. Kodem litery jest adres wierzchołka, do którego przypisana jest ta litera.
Dodatkowo na dole drzewa znajduje się 0-węzeł, w którym znajdują się nie używane dotąd litery alfabetu $\mathcal{A}$ wstępnie zakodowane.

(1) Na początku drzewo składa się z jednego wierzchołka, któremu przyporządkowany jest 0-węzeł, w którym umieszczone są wszystkie litery alfabetu $\mathcal{A}$ wstępnie zakodowane.
(2) Po pobraniu pierwszej litery x kodowanego tekstu, na wyjściu podajemy jej kod w 0-węźle i tworzymy drzewo z własnością rodzeństwa:



(3) Po zakodowaniu n pierwszych liter $x_1, \ldots, x_n$ i stworzeniu drzewa z własnością rodzeństwa pobieramy kolejną literę x_{n+1}.
   (a) Jeśli litera nie była dotychczas używana, to znajduje się w 0-węźle. Jej kodem jest adres 0-węzła+kod litery x_{n+1} w 0 węźle, a nowe drzewo tworzymy z istniejącego drzewa „rozdwajając" 0-węzeł.



   (b) Jeśli litera była już użyta i znajduje się na dotychczasowym drzewie, to jej kod jest adresem węzła na którym się znajduje ta litera, a jej licznik zwiększa się o 1. Jeśli zwiększenie licznika spowoduje zaburzenie własności rodzeństwa, to dokonujemy modyfikacji według zasady opisanej powyżej. Następnie zwiększamy o jeden liczniki wszystkich przodków wierzchołka litery x_{n+1}. Jeśli to znów zaburzy własność rodzeństwa, to kontynuujemy proces modyfikacji aż do chwili dotarcia do korzenia drzewa.

Przykład. Zakodujemy komunikat $aafcccb$. Zakładamy, że w 0-węźle znajdują się litery a, b, c, d, e, f wstępnie zakodowane:

$$\begin{cases} a & \to & 10, \\ b & \to & 110, \\ c & \to & 1110, \\ d & \to & 11110, \\ e & \to & 111110, \\ f & \to & 1111110. \end{cases}$$

Kodowanie arytmetyczne

Zacznijmy od graficznej interpretacji kodowania Shannona.

Mamy dane prawdopodobieństwa $p_1 \geq p_2 \geq \ldots \geq p_n$. Wyznaczamy prawdopodobieństwa kumulatywne $P_1 = 0, P_i = p_1 + \ldots + p_{i-1}$ dla $i > 1$. W szczególności $P_{n+1} = \sum_{i=1}^{n} p_i = 1$.

Dzielimy przedział $[0,1)$ na podprzedziały $[P_i, P_{i+1})$ dla $i \in \{1,\ldots,n\}$, których długości są równe odpowiednio $\Delta_i = P_{i+1} - P_i = p_i$.

Na przykład dla źródła $(\{a,b,c,d,e\}, \{0.3, 0.3, 0.2, 0.1, 0.1\})$ podział jest następujący:

Każdy z przedziałów odpowiada pewnej literze. W naszym przypadku:

W każdy z przedziałów wybieramy element do niego należący, zwany jego **znacznikiem**, który jest liczbowym odpowiednikiem danej litery. W przypadku kodu Shannona, tym znacznikiem jest lewy brzeg przedziału:

Kodem litery a_i jest $\lceil -\log\Delta_i \rceil = \lceil -\log p_i \rceil$ pierwszych cyfr dwójkowego rozwinięcia znacznika przedziału.

Liczbę $\lceil -\log\Delta_i \rceil$ nazywamy **długością Shannona**.

Kodowanie arytmetyczne jest dynamicznym rozwinięciem metody Shannona. W tej metodzie przedział przypisywany jest tekstowi, a nie jak w metodzie Shannona literze.

Dane jest źródło $(\mathcal{A}, P)$ i tekst $v = a_1 a_2 a_3 \ldots$ zapisany w alfabecie $\mathcal{A}$. Zakładamy, że prawdopodobieństwa ułożone sa nierosnąco.

W pierwszym etapie algorytmu tekstowi v przypisujemy przedział zawarty w przedziale $[0,1]$ w następujący sposób.

- Na początku dzielimy przedział $[0,1]$ według opisanej wyżej zasady i wybieramy podprzedział odpowiadający pierwszej literze tekstu.
- Tekstowi odpowiadającemu n pierwszym literom tekstu odpowiada przedział $[L,P)$ o długości $\Delta = P - L$. Przedział ten dzielimy na podprzedziały, których długości odpowiadają kolejnym prawdopodobieństwom (a więc również literom alfabetu

$\mathcal{A}$). Prawdopodobieństwu p_i (literze a_i) odpowiada przedział $[L + \Delta P_{i-1}, L + \Delta P_i)$.

- Pobieramy kolejną literę a_{n+1} i wybieramy ten z podprzedziałów przedziału $[L, P)$, który jej odpowiada.
- Wybór podprzedziałów kontynuujemy aż do wyboru ostatniej litery tekstu. Ten ostatni przedział jest przypisywany kodowanemu tekstowi v.

W wybranym przedziale wybieramy dowolny punkt, który będzie jego znacznikiem. Najczęściej wybieramy albo lewy brzeg przedziału albo jego środek.

Kodem teksu v jest $\lceil -\log\Delta \rceil + 1$ pierwszych cyfr rozwinięcia dwójkowego wybranego przedziału.

Przykład. Dane jest źródło $(\{a, b, c, d\}, \{0.3, 0.3, 0.2, 0.1, 0.1\}$. Zakodować tekst *aab*.

- Dzielimy przedział $[0, 1]$:

  i wybieramy przedział $[0, 0.3)$ odpowiadający literze a.
- Dzielimy przedział $[0, 0.3)$ o długości $\Delta = 0.3$:

  i wybieramy przedział $[0, 0.9)$ odpowiadający literze a.
- Dzielimy przedział $[0, 0.09)$ o długości $\Delta = 0.09$:

  i wybieramy przedział $[0.027, 0.054)$ o długości $\Delta = 0.027$.
- Ponieważ b jest ostatnią literą tekstu, to $[0.027, 0.054)$ jest przedziałem odpowiadającym tekstowi *aab*.
- Jako znacznik przedziału $[0.027, 0.054)$ wybieramy jego środek $\frac{0.027+0.054}{2} = 0.0405$.
- Obliczamy długość Shannona $\lceil -\log 0.027 \rceil + 1 = 7$.

- Wyznaczamy 7 cyfr rozwinięcia binarnego liczby 0.0405:

$$0000101.$$

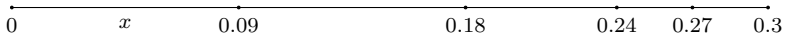
- Kodem tekstu jest 0000101.

Przykład. Dane jest źródło $(\{a, b, c, d\}, \{0.3, 0.3, 0.2, 0.1, 0.1\}$. Odkodować 0000101.

- Ponieważ długość kodu jest równa 7, to długość Δ wybranego przedziału jest zawarta w przedziale $I = \left[\frac{1}{2^6}, \frac{1}{2^5}\right) = [0.015625, 0.03125)$. Będziemy wykonywać kroki algorytmu tak długo aż natrafimy na przedział, którego długość należy do I.
- Kod 0000101 odpowiada liczbie $x = \frac{1}{32} + \frac{1}{128} = 0.0390625$.
- Dzielimy przedział $[0, 1]$:

  i wybieramy przedział $[0, 0.3)$ do którego należy liczba x. Zatem pierwszą literą zakodowanego tekstu była litera a. Ponieważ długość przedziału $[0, 0.3)$ jest równa 0.3 i nie zawiera się w $I = [0.015625, 0.03125)$, to wykonujemy kolejne kroki.

- Dzielimy przedział $[0, 0.3)$ o długości $\Delta = 0.3$:



  i wybieramy przedział $[0, 0.09)$ do którego należy liczba x. Drugą literą jest a. Ponieważ długość przedziału $[0, 0.09)$ jest równa 0.09 i nie zawiera się w $I = [0.015625, 0.03125)$, to wykonujemy kolejne kroki.

- Dzielimy przedział $[0, 0.09)$ o długości $\Delta = 0.09$:

  i wybieramy przedział $[0.027, 0.054)$ do którego należy liczba x. Trzecią literą jest b. Ponieważ długość przedziału $[0.027, 0.054)$ jest równa 0.027 i zawiera się w $I = [0.015625, 0.03125)$, to kończymy proces odkodowania.

- Odkodowanym tekstem jest *aab*.

Po wyborze ostatniego przedziału $[L, P)$ o długości $\Delta = P - L$ musimy wybrać $x = (0, a_1 a_2 \ldots)_2$ tak, aby liczba $x' = (0, a_1 \ldots a_l)_2$, dla $l = \lceil -\log\Delta \rceil + 1$ nadal mieściła się w przedziale $[L, P)$.
Pokażemy, że $x = \frac{L+P}{2}$, jest dobrym wyborem.
Jeśli $2^n \leq \frac{1}{\Delta} \leq 2^{n+1}$, to $l = n + 2$ i $\Delta \geq \frac{1}{2^{n+1}} > \frac{1}{2^{n+2}} = \frac{1}{2^l}$.
Zauważmy, że $x' = x - y$, gdzie $y = (0, \underbrace{0 \ldots 0}_{l} a_{l+1} a_{l+2} \ldots)_2$. Wynika stąd, że $y \leq \frac{1}{2^l}$ i

$$x' = x - y \geq x - \frac{1}{2^l}.$$

Musimy, zatem pokazać, że

$$L \leq x - \frac{1}{2^l}. \tag{1}$$

Ponieważ

$$\frac{1}{2^{l-1}} = \frac{1}{2^{n+1}} \leq \Delta = P - L,$$

to

$$2L + \frac{1}{2^{l-1}} \leq L + P,$$

więc

$$L + \frac{1}{2^l} \leq \frac{L+P}{2}$$

i

$$L \leq \frac{L+P}{2} - \frac{1}{2^l} \leq x'.$$

Zatem dla $x = \frac{L+P}{2}$ nierówność (1) jest spełniona.

Aby uniknąć problemów związanych z porównaniem długości przedziałów, często do alfabetu $\mathcal{A}$ dokłada się znak końca pliku #, któremu przypisuje się pewne prawdopodobieństwo. Wtedy zarówno proces kodowania, jak i dekodowania kończy się wraz z natrafieniem na przedział odpowiadający znakowi #.

Implementacja kodowania arytmetycznego

Jeśli tekst jest długi, to w po pewnym czasie kolejne przedziały są bardzo małe i komputer może nie rozróżniać początku i końca przedziału. Aby uniknąć tego problemu wprowadzono modyfikację algorytmu, która podwaja małe przedziały. Dodatkowo w każdej iteracji koder zwraca kolejne znaki kodu.

Po wybraniu przedziału $I = [L, P)$ wykonujemy następującą pętlę.

Przyjmujemy, że licznik $licznik = 0$.

Dopóki $I \subseteq [0, 0.5)$ lub $I \subseteq [0.5, 1)$ lub $I \subseteq [0.25, 0.75)$.

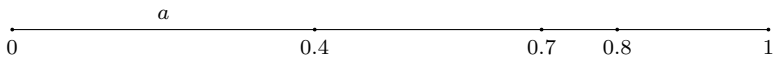
- Jeśli $I \subseteq [0, 0.5)$, to przedział przekształcamy na $[E_1(L), E_1(P))$, gdzie $E_1(x) = 2x$. Koder zwraca $01^{licznik}$ i $licznik = 0$.
- Jeśli $I \subseteq [0.5, 1)$, to przedział przekształcamy na $[E_2(L), E_2(P))$, gdzie $E_2(x) = 2(x - 0.5)$. Koder zwraca $10^{licznik}$ i $licznik = 0$.
- Jeśli $I \subseteq [0.25, 0.75)$, to przedział przekształcamy na $[E_3(L), E_3(P))$, gdzie $E_3(x) = 2(x-0.25)$. Zwiększamy licznik $licznik = licznik+1$.

Po pobraniu ostatniego znaku # wykonujemy powyższa pętlę tak długo aż otrzymamy przedział $I = [a, b)$ nie zawarty w żadnym z powyższych przedziałów.

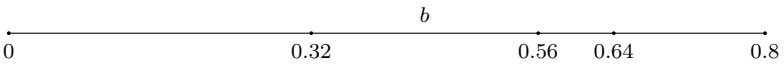
- Jeśli $a < 0.25$ zwracamy $011^{licznik}$,
- Jeśli $a \geq 0.25$ zwracamy $100^{licznik}$.

Przykład. Dane jest źródło $(\{a, b, c, \#\}, \{0.4, 0.3, 0.1, 0.2\})$. Zakodować tekst *abbc#*.

- Dzielimy przedział $[0, 1)$.

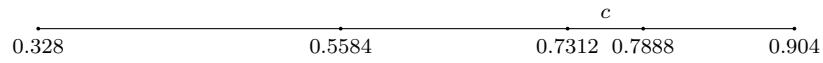


- Wybieramy przedział $I = [0, 0.4)$ odpowiadający pierwszej literze a.
- Ponieważ $I \in [0, 0.5)$ to zwracamy 0 i podwajamy przedział przekształceniem E_1: $I \to [0, 0.8)$. Ten nowy przedział nie zawiera się w żadnym z powyższych przedziałów.
- Dzielimy przedział $[0, 0.8)$:



- Wybieramy przedział $I = [0.32, 0.56)$ odpowiadający drugiej literze tekstu b.
- Ponieważ $I \subseteq [0.25, 0.75)$, to zwiększamy przedział przekształceniem E_3: $I \to I = [0.14, 0.62)$ i zwiększamy $licznik = 1$.
- Ponieważ I nie jest zawarty w żadnym z przedziałów dzielimy go na podprzedziały:

- Wybieramy przedział $I = [0.332, 0.476)$ odpowiadający trzeciej literze tekstu b.
- Ponieważ $I \subseteq [0, 0.5)$, to zwiększamy przedział przekształceniem E_1: $I \to I = [0.664, 0.952)$, zwracamy 01 i $licznik = 0$.
- Ponieważ $I \subseteq [0.5, 1)$, to zwiększamy przedział przekształceniem E_2: $I \to I = [0.328, 0.904)$, zwracamy 1 i $licznik = 0$.
- Ponieważ I nie jest zawarty w żadnym z przedziałów dzielimy go na podprzedziały:



- Wybieramy przedział $I = [0.7312, 0.7888)$ odpowiadający czwartej literze tekstu c.
- Ponieważ $I \subseteq [0.5, 1)$, to zwiększamy przedział przekształceniem E_2: $I \to I = [0.4624, 0.5776)$, zwracamy 1 i $licznik = 0$.
- Ponieważ $I \subseteq [0.25, 0.75)$, to zwiększamy przedział przekształceniem E_3: $I \to I = [0.4248, 0, 6552)$, $licznik = 1$.
- Ponieważ $I \subseteq [0.25, 0.75)$, to zwiększamy przedział przekształceniem E_3: $I \to I = [0.3496, 0.8104)$, $licznik = 2$.
- Ponieważ I nie jest zawarty w żadnym z przedziałów dzielimy go na podprzedziały:

- Wybieramy przedział $I = [0.71824, 0.8104)$ odpowiadający ostatniej literze tekstu #.
- Ponieważ $I \subseteq [0.5, 1)$, to zwiększamy przedział przekształceniem E_2: $I \to I = [0.43648, 0.6208)$, zwracamy 100 i $licznik = 0$.
- Ponieważ $I \subseteq [0.25, 0.75)$, to zwiększamy przedział przekształceniem E_3: $I \to I = [0.37296, 0.7416)$, $licznik = 1$.
- Ponieważ $I \subseteq [0.25, 0.75)$, to zwiększamy przedział przekształceniem E_3: $I \to I = [0.24592, 0.9832)$, $licznik = 2$.
- Ponieważ $I = [0.24592, 0.9832)$ nie jest zawarty w żadnym z podprzedziałów i $0.24592 < 0.25$, to zwracamy 0111 i kończymy kodowanie.

- Kodem przedziału jest 001111000111.

Informacja i kompresja danych
Wykład 4

Kody słownikowe

Kod LZ77

W metodzie LZ77 używamy bufora, który składa się z kodowanych liter tekstu. Bufor składa się z dwóch części. Część pierwsza o zadanej długości l_1 nazywa się *buforem słownika* i zawiera l_1 ostatnio zakodowanych liter tekstu, a część druga o zadanej długości l_2 nazywa się *buforem kodowania* i zawiera l_2 kolejnych liter tekstu.

Na początku zapełniamy bufor słownikowy l_1 kopiami pierwszej litery kodowanego tekstu.

W każdej iteracji szukamy w buforze słownikowym najdłuższego (jednakże nie dłuższego niż $l_1 - 1$) podciągu pc równego początkowi ciągu umieszczonego aktualnie w buforze kodowania.

Jeśli taki podciąg pc zostanie znaleziony, to na wyjściu podajemy trójkę (p, l, s) taką, że p jest liczbą określającą pozycję początkową ciągu pc w buforze słownikowym, l jest długością ciągu pc, a s pierwszą literą występującą po ciągu pc w buforze kodowania.

Następnie bufor przesuwany jest o $l+1$ pozycji w lewo o wolne miejsca zapełniane są kolejnymi literami kodowanego tekstu.

Przykład. Zakładamy, że $l_1 = l_2 = 4$ pozycje bufora numerujemy liczbami $0, \ldots, 7$. Zakodujemy tekst $aababacbaacbaadaaa$. Symbole BS i BK oznaczają bufory słownikowy i kodowania.

$l+1$	BS				BK				Wyjście
	0	1	2	3	4	5	6	7	
	a	a	a	a					a
	$\|aaba\|bacbaacbaadaaa$								
	a	a	a	a	a	a	b	a	$22b$
3	$aab\|abac\|baacbaadaaa$								
	a	a	a	b	a	b	a	c	$23c$
4	$aababac\|baac\|baadaaa$								
	a	b	a	c	b	a	a	c	$12a$
3	$aababacbaa\|cbaa\|daaa$								
	c	b	a	a	c	b	a	a	$03a$
4	$aababacbaacbaa\|daaa\|$								
	c	b	a	a	d	a	a	a	$00d$
1	$aababacbaacbaad\|aaa\|$								
	b	a	a	d	a	a	a		$12a$

Otrzymanym kodem jest

$$a, 22b, 23c, 12a, 03a, 30d, 12a.$$

Kod LZ78

W metodzie LZ78 tworzony jest na bieżąco słownik, w którym przechowywane są dotychczas pojawiające się frazy. Kolejne hasła dopisywane są do słownika pod kolejnymi numerami.

Na wyjściu podawane są pary (p, s) takie, że p oznacza pozycję ciągu w słowniku, a s symbol następujący na wyjściu po dopasowanym ciągu.

Przykład. Zakodujemy tekst $aababacbaacbaadaaa$.

Koder		Słownik	
wejście	wyjście	numer	hasło
$aababacbaacbaadaaa$			
a	$0a$	1	a
$aababacbaacbaadaaa$			
ab	$1b$	2	ab
$aababacbaacbaadaaa$			
aba	$2a$	3	aba
$aababacbaacbaadaaa$			
c	$0c$	4	c
$aababacbaacbaadaaa$			
b	$0b$	5	b
$aababacbaacbaadaaa$			
aa	$1a$	6	aa
$aababacbaacbaadaaa$			
cb	$4b$	7	cb
$aababacbaacbaadaaa$			
aad	$6d$	8	aad
$aababacbaacbaadaaa$			
aaa	$6a$	9	aaa

Otrzymanym kodem jest:

$$0a, 1b, 2a, 0c, 0b, 1a, 4b, 6d, 6a.$$

Kod LZW

W metodzie LZW wstępnie tworzy się słownik złożony z liter użytych w tekście. Na wyjściu podaje się tylko jedną wartość: pozycję dopasowanego ciągu w słowniku, a kolejne litery w buforze kodowania przesuwa się o jedna pozycję mniej.

Przykład. Zakodujemy tekst *aababacbaacbaadaaa*.

Koder		Słownik	
wejście	wyjście	numer	hasło
		1	*a*
		2	*b*
		3	*c*
		4	*d*
aa	1	5	*aa*
ab	1	6	*ab*
ba	2	7	*ba*
aba	6	8	*aba*
ac	1	9	*ac*
cb	3	10	*cb*
baa	7	11	*baa*
acb	9	12	*acb*
baad	11	13	*baad*
da	4	14	*da*
aaa	5	15	*aaa*

Otrzymanym kodem jest:

$$1, 1, 2, 6, 1, 3, 7, 9, 11, 4, 5.$$

Kodowanie liczb naturalnych

Kod unarny

Kodem liczby k jest ciąg złożony z k jedynek zakończony zerem, to znaczy $k \to \underbrace{1 \ldots 1}_{k}0$. Na przykład $5 \to 111110$.

Kod reszt modulo m

Wybieramy liczbę m oraz liczbę całkowitą $r \in \{0, \ldots, m-1\}$.

Oznaczamy przez $B_n(k)$ n-bitowy zapis liczby k.

Określamy wartość $n = \lfloor \log m \rfloor$.

Określamy wartość $p = 2^{\lceil \log m \rceil} - m$

- Jeśli $r < p$, to jej kodem jest $B_n(r)$.
- Jeśli $r \geq p$, to jej kodem jest $B_{n+1}(r + p)$.

Przykład. Przyjmujemy $m = 5$. Zatem kodować będziemy liczby $0, 1, 2, 3, 4$.

- Obliczamy $n = \lfloor \log 5 \rfloor = 2$.
- Obliczamy $p = 2^{\lceil \log 5 \rceil} - 5 = 2^3 - 5 = 3$.
- Liczby $0, 1, 2$ są mniejsze od $p = 3$, więc kodujemy $0, 1, 2$ na $n = 2$ bitach, a $3, 4$ są większe bądź równe od $p = 3$, więc ich kodami są rozwinięcia binarne liczb $3 + 3, 4 + 3$ na $n + 1 = 3$ bitach.
- Kodujemy

$$\begin{aligned} 0 &\to 0 \to 00, \\ 1 &\to 1 \to 01, \\ 2 &\to 2 \to 10, \\ 3 &\to 6 \to 110, \\ 4 &\to 7 \to 111. \end{aligned}$$

Kod Golomba

Wybieramy m.

Kodować będziemy liczbę całkowitą k.

Dzielimy k przez m z resztą:

$$k = qm + r, \ 0 \leq r < m,$$

gdzie $q = \lfloor \frac{k}{m} \rfloor$.

Liczbę q kodujemy kodem unarnym, r kodem reszt modulo n.

Przykład. Przyjmujemy $m = 5$. Zakodujemy 28.

Ponieważ $28 = 5 \cdot 5 + 3$, to 5 kodujemy na 111110, a r na 110. Zatem kodem liczby 28 jest

$$28 \to 111110110.$$

Kod Tunstalla

Wszystkie słowa kodowe maja tą samą długość ale jeden kod może kodować różną liczbę liter alfabetu wejściowego.

Dane jest źródło $(\mathcal{A} = \{a_1, a_2, \ldots, a_n\}, \{p_1, \ldots, p_n\})$.

Ustalamy długość słów kodowych m.

- Przyporządkowujemy symbolom alfabetu m różnych słów kodowych o długości m).
- Dopóki liczba niewykorzystanych słów kodowych jest większa niż $N-1$:
  - wybieramy słowo kodowe x odpowiadające ciągowi o największym prawdopodobieństwie;
  - usuwamy x z kodu;
  - dodajemy do kodu ciągi powstałe z dodania $a_1, \ldots, a_n$ jako sufiksów ciągu odpowiadającego kodowi x i przypisujemy im odpowiednie prawdopodobieństwa.

Co najmniej jedno słowo kodowe będzie niewykorzystane. To słowo będzie wykorzystywane do rozdzielenie partii tekstu, która odpowiada kodowanym słowom od parti tekstu, których nie ma wśród słów kodowych.

Przykład. Dane jest źródło $(\{a, b, c\}, \{0.6, 0.3, 0.1\})$. Zatem $n = 3$.

Wstępnie tworzymy kod liter kod : $a \to 00,\ b \to 01,\ c \to 10$.

Ustalamy $m = 3$. Zatem liczba słów kodowych jest równa $2^m = 8$.

Kroki algorytmu kontynuujemy tak długo, aż liczba niewykorzystanych słów kodowych jest większa od $n - 1 = 2$.

- Usuwamy literę a o największym prawdopodobieństwie 0.6 i tworzymy nowe źródło

$$\{aa, ab, ac, b, c\}, \{0.36, 0.18, 0.06, 0.3, 0.1\}$$

- Usuwamy ciąg aa o największym prawdopodobieństwie a i tworzymy nowe źródło

$$\{aaa, aab, aac, ab, ac, b, c\}, \{0.216, 0.108, 0.036, 0.18, 0.06, 0.3, 0.1\}$$

- Ponieważ liczba otrzymanych ciągów jest równa 7, a słów kodowych długości 3 jest 8, to liczba niewykorzystanych ciągów jest równa $8 - 7 < n - 1 = 2$. Zatem możemy przystąpić do kodowania:

aaa	aab	aac	ab	ac	b	c	-
000	001	010	011	100	101	110	111

- Zakodujemy tekst $abcaabbaa$.
- Dzielimy go na części odpowiadające słowom kodowym $ab|c|aab|b|aa$.

- Kodujemy 011|110|001|101|kod rozdzielający|kod(a)kod(a).
- 011|110|001|101|111|00|00.

RLE (Run-Length Encoding - kodowanie długości serii)

Prosta metoda bezstratnej kompresji danych, której działanie polega na opisywaniu ciągów tych samych liter za pomocą licznika powtórzeń (długości serii), a dokładniej przez pary: $licznik_powtrze_litery$, $litera$.

Przykład. Ciąg $aaaabbbbbccccccc$ zakodujemy jako $4a5b6c$.

Informacja i kompresja danych
Wykład 5

Kodowanie liczb naturalnych

Kod unarny

Kodem liczby k jest ciąg złożony z k jedynek zakończony zerem, to znaczy $k \to \underbrace{1 \ldots 1}_{k} 0$. Na przykład $5 \to 111110$.

Kod reszt modulo m

Wybieramy liczbę m oraz liczbę całkowitą $r \in \{0, \ldots, m-1\}$.

Oznaczamy przez $B_n(k)$ n-bitowy zapis liczby k.

Określamy wartość $n = \lfloor \log m \rfloor$.

Określamy wartość $p = 2^{\lceil \log m \rceil} - m$

- Jeśli $r < p$, to jej kodem jest $B_n(r)$.
- Jeśli $r \geq p$, to jej kodem jest $B_{n+1}(r+p)$.

Przykład. Przyjmujemy $m = 5$. Zatem kodować będziemy liczby $0, 1, 2, 3, 4$.

- Obliczamy $n = \lfloor \log 5 \rfloor = 2$.
- Obliczamy $p = 2^{\lceil \log 5 \rceil} - 5 = 2^3 - 5 = 3$.
- Liczby $0, 1, 2$ są mniejsze od $p = 3$, więc kodujemy $0, 1, 2$ na $n = 2$ bitach, a $3, 4$ są większe bądź równe od $p = 3$, więc ich kodami są rozwinięcia binarne liczb $3 + 3, 4 + 3$ na $n + 1 = 3$ bitach.
- Kodujemy

$$\begin{aligned} 0 &\to 0 \to 00, \\ 1 &\to 1 \to 01, \\ 2 &\to 2 \to 10, \\ 3 &\to 6 \to 110, \\ 4 &\to 7 \to 111. \end{aligned}$$

Kod Golomba

Wybieramy m.

Kodować będziemy liczbę całkowitą k.

Dzielimy k przez m z resztą:

$$k = qm + r,\ 0 \leq r < m,$$

gdzie $q = \lfloor \frac{k}{m} \rfloor$.

Liczbę q kodujemy kodem unarnym, r kodem reszt modulo n.

Przykład. Przyjmujemy $m = 5$. Zakodujemy 28.

Ponieważ $28 = 5 \cdot 5 + 3$, to 5 kodujemy na 111110, a r na 110. Zatem kodem liczby 28 jest

$$28 \to 111110110.$$

Kod Tunstalla

Wszystkie słowa kodowe maja tą samą długość ale jedno słowo kodowe może kodować różną liczbę liter alfabetu wejściowego.

Dane jest źródło $(\mathcal{A} = \{a_1, a_2, \ldots, a_n\}, \{p_1, \ldots, p_n\})$.

Ustalamy długość słów kodowych m.

- Przyporządkowujemy symbolom alfabetu n różnych słów kodowych o takiej samej długości.
- Dopóki liczba niewykorzystanych słów kodowych długości m jest większa niż $n - 1$:
  - wybieramy słowo kodowe x odpowiadające ciągowi o największym prawdopodobieństwie;
  - usuwamy x z kodu;
  - dodajemy do kodu ciągi powstałe z dodania $a_1, \ldots, a_n$ jako sufiksów ciągu odpowiadającego kodowi x i przypisujemy im odpowiednie prawdopodobieństwa.

Co najmniej jedno słowo kodowe będzie niewykorzystane. To słowo będzie wykorzystywane do rozdzielenie partii tekstu, która odpowiada kodowanym słowom od partii tekstu, których nie ma wśród słów kodowych.

Przykład. Dane jest źródło $(\{a, b, c\}, \{0.6, 0.3, 0.1\})$. Zatem $n = 3$.

Wstępnie tworzymy kod liter kod : $a \to 00,\ b \to 01,\ c \to 10$.

Ustalamy $m = 3$. Zatem liczba słów kodowych jest równa $2^m = 8$.

Kroki algorytmu kontynuujemy tak długo, aż liczba niewykorzystanych słów kodowych jest większa od $n - 1 = 2$.

- Usuwamy literę a o największym prawdopodobieństwie 0.6 i tworzymy nowe źródło

$$\{aa, ab, ac, b, c\}, \{0.36, 0.18, 0.06, 0.3, 0.1\}$$

- Usuwamy ciąg aa o największym prawdopodobieństwie a i tworzymy nowe źródło

$$\{aaa, aab, aac, ab, ac, b, c\}, \{0.216, 0.108, 0.036, 0.18, 0.06, 0.3, 0.1\}$$

- Ponieważ liczba otrzymanych ciągów jest równa 7, a słów kodowych długości 3 jest 8, to liczba niewykorzystanych ciągów jest równa $8 - 7 < n - 1 = 2$. Zatem możemy przystąpić do kodowania:

aaa	aab	aac	ab	ac	b	c	-
000	001	010	011	100	101	110	111

- Zakodujemy tekst $abcaabbaa$.
- Dzielimy go na części odpowiadające słowom kodowym $ab|c|aab|b|aa$.
- Kodujemy 011|110|001|101|kod rozdzielający|kod(a)kod(a).
- 011|110|001|101|111|00|00.

RLE (Run-Length Encoding - kodowanie długości serii)

Prosta metoda bezstratnej kompresji danych, której działanie polega na opisywaniu ciągów tych samych liter za pomocą licznika powtórzeń (długości serii), a dokładniej przez pary: $licznik\,powtorzen\,litery$, $litera$.

Przykład. Ciąg $aaaabbbbbccccccc$ zakodujemy jako $4a5b6c$.

Transformata Burrowsa-Wheelera (BWT)

Transformata Burrowsa-Wheelera ma na celu przekształcenie tekstu $x_1 \dots x_n$ na tekst, w którym litery zostaną przemieszczone tak aby pojawiał się w nim jak największa liczba powtórzeń. Tekst, który otrzymujemy ma taką samą długość, a poszczególne litery występują w nim taką samą liczbę razy jak w tekście oryginalnym. W celu zastosowania tego algorytmu należy znać cały tekst, a nie jak na przykład w algorytmie arytmetycznym kolejne litery.

Po zastosowaniu BWT tekst kodujemy algorytmem, który uwzględnia powtórzenia liter. Na przykład move-to-front, opisany poniżej.

Algorytm BTW

Tekstem, który przetwarzamy jest $x_1 \dots x_n$.

1. Tworzymy tablicę wymiaru $n \times n$ złożoną z n cyklicznych przesunięć tekstu $x_1 \dots x_n$:

$$\begin{array}{cccccc} x_1 & x_2 & x_3 & \dots & x_{n-1} & x_n \\ x_2 & x_3 & x_4 & \dots & x_n & x_1 \\ x_3 & x_4 & x_5 & \dots & x_1 & x_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n-1} & x_n & x_1 & \dots & x_{n-3} & x_{n-2} \\ x_n & x_1 & x_2 & \dots & x_{n-2} & x_{n-1} \end{array}$$

2. Porządkujemy otrzymaną tablicę leksykograficznie (to znaczy w porządku alfabetycznym), otrzymując nową tablicę A.
3. Na wyjściu podajemy ostatnią kolumnę tablicy A, którą oznaczamy przez S oraz numer N wiersza, w którym znajduje się oryginalny tekst w tej tablicy.

Przykład. Dany jest tekst *abacbdaaebc*, o długości 11.

Tworzymy tablicę przesunięć cyklicznych

0	a	b	a	c	b	d	a	a	e	b	c
1	b	a	c	b	d	a	a	e	b	c	a
2	a	c	b	d	a	a	e	b	c	a	b
3	c	b	d	a	a	e	b	c	a	b	a
4	b	d	a	a	e	b	c	a	b	a	c
5	d	a	a	e	b	c	a	b	a	c	b
6	a	a	e	b	c	a	b	a	c	b	d
7	a	e	b	c	a	b	a	c	b	d	a
8	e	b	c	a	b	a	c	b	d	a	a
9	b	c	a	b	a	c	b	d	a	a	e
10	c	a	b	a	c	b	d	a	a	e	b

Porządkujemy tablicę leksykograficznie, otrzymując tablicę A:

0	a	a	e	b	c	a	b	a	c	b	d
1	a	b	a	c	b	d	a	a	e	b	c
2	a	c	b	d	a	a	e	b	c	a	b
3	a	e	b	c	a	b	a	c	b	d	a
4	b	a	c	b	d	a	a	e	b	c	a
5	b	c	a	b	a	c	b	d	a	a	e
6	b	d	a	a	e	b	c	a	b	a	c
7	c	a	b	a	c	b	d	a	a	e	b
8	c	b	d	a	a	e	b	c	a	b	a
9	d	a	a	e	b	c	a	b	a	c	b
10	e	b	c	a	b	a	c	b	d	a	a

Na wyjściu podajemy ostatnią kolumnę S : *dcbaecbaba* i pozycję oryginalnego tekstu $N = 1$.

Procedura odzyskania oryginalnego tekstu polega na odtworzeniu tablicy A i wykorzystuje numer pozycji tekstu w tej tablicy.

W trakcie tej procedury będziemy tworzyć permutację T taką, że $T(i)$ jest numerem wiersza w tabeli A, w którym występuje ciąg powstający z ciągu w wierszu i przez przestawienie ostatniego znaku na początek. W naszym przykładzie tę permutację widać w dwóch pierwszych kolumnach ostatniej tabeli:

$$T = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 9 & 7 & 4 & 0 & 1 & 10 & 8 & 5 & 2 & 6 & 3 \end{pmatrix} = [9\ 7\ 4\ 0\ 1\ 10\ 8\ 5\ 2\ 6\ 3].$$

Określamy też funkcje F i L. $F(i)$ jest pierwszą literą ciągu w wierszu i, $L(i)$ ostatnią. Na przykład $F(2) = a$, $L(2) = b$.
Z określenia funkcji T wynika, że $F(T(i)) = L(i)$.
Jest jasne, że $L(i)$ poprzedza cyklicznie $F(i)$, to znaczy, że za literą o numerze $L(i)$ jest litera $F(i)$ (być może, że $L(i)$ jest na końcu tekstu, a $F(i)$ na początku).
Zatem $L(T(i))$ poprzedza $F(T(i)) = L(i)$.
Korzystając z poprzedniej uwagi możemy wyznaczyć oryginalny ciąg od ostatniej litery do pierwszej.
Ostatnią literą tekstu jest $L(1) = c$, wtedy przedostatnią jest $L(T(1)) = L(7) = b$, a kolejną $L(T(7)) = L(5) = e$. Poniższa tabela pokazuje kolejne kroki:

$L(1)$	c
$L(T(1)) = L(7)$	b
$L(T(7)) = L(5)$	e
$L(T(5)) = L(10)$	a
$L(T(10)) = L(3)$	a
$L(T(3)) = L(0)$	d
$L(T(0)) = L(9)$	b
$L(T(9)) = L(6)$	c
$L(T(6)) = L(8)$	a
$L(T(8)) = L(2)$	b
$L(T(2)) = L(4)$	a

Oryginalny tekst odczytujemy od dołu do góry *abacbdaaeb*.

Pozostaje jeszcze sposób wyznaczania permutacji T odczytujemy ją przy pomocy funkcji F i L. Jeśli w kolumnie L znajdują się te same litery, to odczytujemy w takiej kolejności jak w kolumnie F:

n	F	L	Numer litery $L(n)$ w kolumnie F
0	a	d	9
1	a	c	7
2	a	b	4
3	a	a	0
4	b	a	1
5	b	e	10
6	b	c	8
7	c	b	5
8	c	a	2
9	d	b	6
10	e	a	3

Stąd otrzymujemy $T = [9\ 7\ 4\ 0\ 1\ 10\ 8\ 5\ 2\ 6\ 3]$.

Kodowanie move-to-front

Litery alfabetu numerujemy liczbami $0, 1, 2, \ldots$. Kodem litery jest jej numer, a następnie przesuwamy ją na początek alfabetu. Zatem jeśli litera się powtarza, to kolejnym jej kodem jest 0.