MINUT 2026 (8), s. 38–52
ISSN 2719-3063

MATEMATYKA I INFORMATYKA NA UCZELNIACH TECHNICZNYCH

Wiktor JAROSZ[1]

[1]Student, Wydział Matematyki Stosowanej, Politechnika Śląska, ul. Kaszubska 23, 44-100 Gliwice

Kodowanie arytmetyczne bez tajemnic: od teorii do praktycznej implementacji

Streszczenie. W artykule omówiono sposób działania kodowania arytmetycznego oraz przedstawiono teorię matematyczną zapewniającą jego poprawność. Ważnym elementem są również ilustracje, które pokazują intuicję stojącą za algorytmem. Szczegółowo opisany został mechanizm renormalizacji — jedna z najbardziej kluczowych części implementacji kodowania arytmetycznego. Artykuł przechodzi również przez przykład kodowania i dekodowania wiadomości oraz omawia standardowy sposób implementacji programowej.

Słowa kluczowe: teoria informacji, kompresja danych, kodowanie arytmetyczne

1. Wstęp

Choć wielu nie zdaje sobie z tego sprawy, bezstratna kompresja danych towarzyszy nam na każdym kroku w życiu cyfrowym. Algorytmy kompresji wykorzystywane są przy praktycznie każdej wizycie na stronach internetowych. Popularne formaty danych jak PNG czy PDF zawierają w sobie wyszukane metody kompresji, które sprawiają, że nasze pliki zajmują znacznie mniej miejsca na dysku, niż gdybyśmy przechowywali je w „surowej" formie. Wśród dziesiątek popularnych metod kompresji bezstratnej, jedna wyróżnia się niebywałą matematyczną pomysłowością, która pozwala na uzyskanie wydajności niezwykle bliskiej teoretycznej granicy. Mowa oczywiście o kodowaniu arytmetycznym.

2. Wprowadzenie do kodowania i kompresji danych

Celem kompresji danych jest redukcja objętości informacji przy zachowaniu jej treści. W przypadku metod bezstratnych, proces ten musi być w pełni odwracalny, co oznacza, że po dekompresji otrzymujemy dokładnie ten sam ciąg bitów, co na wejściu.

Fundamentalnym modelem w teorii informacji jest dyskretne źródło bezpamięciowe (DMS, ang. Discrete Memoryless Source), oznaczane jako zmienna losowa X. Zmienna ta przyjmuje wartości ze skończonego alfabetu $S = \{s_1, s_2, \ldots, s_n\}$ zgodnie z rozkładem prawdopodobieństwa $P = \{p_1, p_2, \ldots, p_n\}$.

Autor korespondencyjny: W. Jarosz (wj316047@student.polsl.pl).
Data wpłynięcia: 18.03.2026.

Intuicja stojąca za efektywną kompresją opiera się na obserwacji, że symbole nie występują w danych z jednakową częstością. Metody statystyczne, takie jak kodowanie Huffmana czy arytmetyczne, przypisują krótsze ciągi bitowe symbolom częstym (o wysokim p_i), a dłuższe symbolom rzadkim.

Miara „losowości” symboli danego źródła nazywana jest entropią Shannona $H(X)$ i wyraża się wzorem:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i). \tag{1}$$

Im bardziej niepewna jest odpowiedź na pytanie „Jaki prawdopodobnie będzie następny symbol?”, tym wyższa jest entropia źródła. Na przykład, wiadomość *acbcab* ma wyższą entropię, niż wiadomość *aaaaab*. Zgodnie z twierdzeniem Shannona o kodowaniu źródła [3], entropia wyznacza teoretyczną dolną granicę średniej długości kodu jednego znaku. Żaden bezstratny algorytm kompresji nie może (średnio) zakodować danych używając mniejszej liczby bitów na symbol, niż wynosi entropia źródła.

3. Kodowanie arytmetyczne

Kodowanie arytmetyczne jest jedną z najbardziej efektywnych metod kompresji bezstratnej. Wiele osób błędnie utożsamia kod Huffmana [1] z granicą kompresji. W praktyce ma on jednak poważne ograniczenie — każdy symbol alfabetu kodowany jest na całkowitej liczbie bitów. Sprawia to, że w skrajnych przypadkach, kod Huffmana może marnować średnio prawie 1 bit dla każdego symbolu. Jest to szczególnie widoczne w następującym przykładzie:

Przykład 1. Obliczmy entropię oraz kod Huffmana źródła X o alfabecie $S = \{a, b\}$ i rozkładzie prawdopodobieństwa $P(a) = 0{,}999$ oraz $P(b) = 0{,}001$.

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i) = -(0{,}999 \log_2(0{,}999) + 0{,}001 \log_2(0{,}001)) \approx 0{,}011 \text{ bita.} \tag{2}$$

Przykładowy kod Huffmana to:

$$a \to 0,$$
$$b \to 1.$$

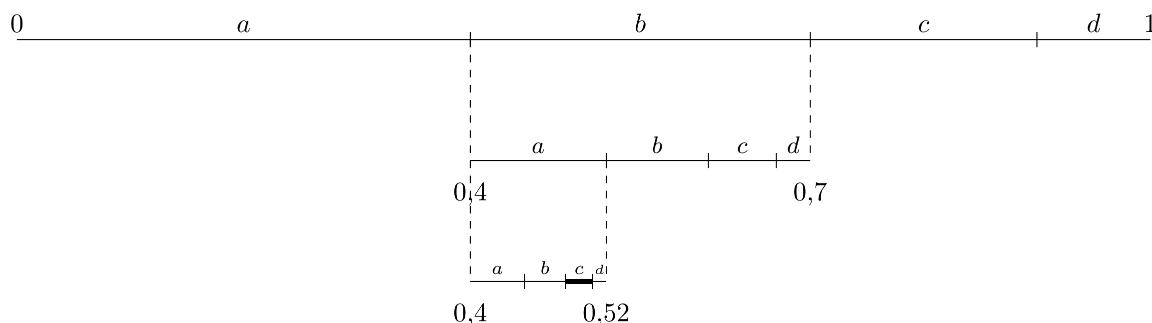
Jak widać, entropia źródła (teoretyczna minimalna średnia długość jednego znaku) wynosi zaledwie 0,011 bita. Jednak używając kodu Huffmana, jesteśmy zmuszeni do przypisania symbolowi a (oraz b) jednego bitu. W tym wypadku, wiadomość złożona z n znaków zostanie zapisana na n bitach — dużo więcej niż teoretyczna granica, która wynosi $0{,}011n$.

Problem ten rozwiązuje kodowanie arytmetyczne, które nie przypisuje słów kodowych pojedynczym symbolom. Zamiast tego, koduje ono cały ciąg wejściowy jako jedną liczbę, co statystycznie pozwala na zapisywanie symboli na „ułamkach” bitów.

3.1. Opis działania kodu

Zasada działania kodera arytmetycznego opiera się na rekurencyjnym dzieleniu przedziału liczbowego. Początkowym przedziałem jest $[0; 1)$. Przedział ten dzielony jest proporcjonalnie do prawdopodobieństw

symboli. Z otrzymanych podprzedziałów wybierany jest podprzedział odpowiadający aktualnie kodowanemu symbolowi. Procedura ta jest powtarzana aż do zakodowania całej wiadomości. Rysunek 1 ilustruje intuicję stojącą za działaniem kodera.



Rysunek 1. Graficzna reprezentacja procesu kodowania ciągu „bac" dla źródła z rozkładem prawdopodobieństwa $\{0{,}4; 0{,}3; 0{,}2; 0{,}1\}$. Każdy kolejny poziom ilustruje fizyczne zawężanie przedziału kodu. Ostatecznym kodem wiadomości jest jakakolwiek liczba z przedziału $[0{,}484; 0{,}508)$.

Niech dany będzie alfabet $S = \{s_1, s_2, \ldots, s_n\}$ oraz stowarzyszony z nim rozkład prawdopodobieństwa $P = \{p_1, p_2, \ldots, p_n\}$. Aby precyzyjnie określić granice podprzedziałów dla każdego symbolu, niezbędne jest wprowadzenie definicji prawdopodobieństwa skumulowanego.

Definicja 1. *Niech C_k będzie sumą prawdopodobieństw symboli $s_1, s_2, \ldots, s_{k-1}$. C_k nazywamy prawdopodobieństwem skumulowanym k-tego symbolu alfabetu S:*

$$C_k = \sum_{i=1}^{k-1} p_i \quad \text{dla } k > 1, \quad \text{oraz } C_1 = 0. \tag{3}$$

Przykład 2. Dla źródła o rozkładzie prawdopodobieństwa $P = \{0{,}4; 0{,}3; 0{,}2; 0{,}1\}$, prawdopodobieństwa skumulowane to:

$$\begin{aligned} C_1 &= 0; \\ C_2 &= 0{,}4; \\ C_3 &= 0{,}7; \\ C_4 &= 0{,}9. \end{aligned}$$

Algorytm utrzymuje dwie zmienne stanu: L (low) reprezentującą dolną granicę aktualnego przedziału oraz H (high) reprezentującą górną granicę. Początkowo $L = 0$ oraz $H = 1$, więc przedziałem jest $[0; 1)$. Proces kodowania kolejnego symbolu polega na zawężaniu przedziału $[L; H)$ do nowego przedziału odpowiadającego temu symbolowi. Operację kodowania k-tego symbolu opisują wzory rekurencyjne:

$$\text{szerokość} = H_{stare} - L_{stare}, \tag{4}$$

$$L_{nowe} = L_{stare} + \text{szerokość} \times C_k, \tag{5}$$

$$H_{nowe} = L_{stare} + \text{szerokość} \times (C_k + p_k). \tag{6}$$

Kodem wiadomości jest jakakolwiek liczba rzeczywista znajdująca się w przedziale końcowym $[L_n; H_n)$. Każdy kolejny krok algorytmu sprawia, że przedział staje się węższy. Im węższy stanie się przedział, tym większa liczba cyfr (bitów) jest potrzebna do zapisania liczby, która się w nim zawiera. Dzięki temu, kodowanie arytmetyczne „zapisuje" symbole o wysokich prawdopodobieństwach na mniejszej liczbie bitów. Symbole takie mają mniejszy wpływ na szerokość przedziału. Z drugiej strony, symbol o prawdopodobieństwie 0,01 zmniejszy przedział 100-krotnie.

Dekodowanie wiadomości odbywa się analogicznie do kodowania. Dekoder otrzymuje kod (liczbę z przedziału) oraz źródło (alfabet i rozkład prawdopodobieństwa). Warto zauważyć, że każdy następny przedział zawiera się w poprzednim: $[L_n; H_n) \subseteq [L_{n-1}; H_{n-1})$. Oznacza to, że jeżeli dekoder otrzyma liczbę 0,242, a pierwszy przedział $[0; 1)$ zawiera podprzedział $[0; 0{,}4)$ odpowiadający symbolowi 'a', to dekoder wypisze właśnie symbol 'a'. Wybranie jakiegokolwiek innego podprzedziału (np. $[0{,}4; 0{,}7)$) nie pozwoliłoby mu na dojście do liczby 0,242.

3.2. Zakończenie kodowania

W praktyce konieczne jest poinformowanie dekodera o końcu wiadomości. Zauważmy, że liczba 0,0 może być kodem wiadomości *a*, ale również *aa, aaa, aaaa, ...* . Istnieje kilka sposobów na rozwiązanie tej dwuznaczności. Jednym z nich jest przesyłanie liczby znaków, które dekoder powinien odczytać. To podejście sprawdza się jednak tylko wtedy, gdy z góry znamy rozmiar przesyłanych danych, co często jest niemożliwe w przypadku transmisji strumieniowej. Zamiast tego, jedna z najpopularniejszych implementacji kodu arytmetycznego [4] używa specjalnego symbolu EOF (ang. End Of File).

Definicja 2. *Niech $\#$ będzie symbolem oznaczającym koniec wiadomości. Wtedy $\# \in S$ z pewnym prawdopodobieństwem $P(\#)$. Koder kończy kodowanie wiadomości symbolem $\#$. Napotkanie symbolu $\#$ przez dekoder oznacza, że cała wiadomość została już odkodowana.*

Symbol $\#$ jest dodawany do alfabetu S z pewnym niskim prawdopodobieństwem $P(\#)$, np. 0,1. Prawdopodobieństwa wystąpienia reszty symboli muszą zostać znormalizowane tak, aby suma wszystkich prawdopodobieństw wynosiła 1.

3.3. Renormalizacja

Przedstawiony dotychczas model kodowania zakładał wykorzystanie maszyny zdolnej do operowania na liczbach rzeczywistych o nieskończonej precyzji. Jednak w rzeczywistości, dokładność liczb jest ograniczona. Ponieważ szerokość przedziału maleje wykładniczo wraz z każdym znakiem, nietrudno zauważyć, że przedział ten szybko by „zniknął". Dolna granica L i górna granica H znalazłyby się tak blisko siebie, że procesor nie byłby w stanie ich rozróżnić, tzw. „underflow". Rozwiązaniem tego problemu jest **renormalizacja**.

Idea działania renormalizacji opiera się na obserwacji, że pewne bity kodu są nam znane jeszcze przed obliczeniem ostatecznego przedziału. Jeżeli podprzedziałem pierwszego symbolu będzie $[0; 0{,}4)$, to może-

my być pewni, że kod wiadomości zacznie się od bitu 0. Jest tak dlatego, że binarna reprezentacja każdej liczby w tym przedziale zaczyna się właśnie od $0{,}0\ldots$. Analogicznie, jeżeli wybranym podprzedziałem będzie $[0{,}7; 0{,}9)$, to będziemy pewni, że następnym bitem kodu wiadomości będzie 1, ponieważ binarna reprezentacja każdej liczby $0{,}7 \leq x < 0{,}9$ zaczyna się od $0{,}1\ldots$. W takiej sytuacji możemy zapisać znany bit do bufora (lub wysłać go do odbiorcy) i „rozciągnąć” przedział poprzez przesunięcie wszystkich bitów w lewo.

Formalnie, po wybraniu podprzedziału $[L; H)$ następnego kodowanego symbolu, mamy trzy przypadki wymagające renormalizacji:

1. Podprzedział zawiera się w pełni w dolnej połowie przedziału: $[L; H) \subseteq [0; 0{,}5)$.

2. Podprzedział zawiera się w pełni w górnej połowie przedziału: $[L; H) \subseteq [0{,}5; 1{,}0)$.

3. Podprzedział zawiera się na pograniczu dolnej oraz górnej połowy przedziału: $[L; H) \subseteq [0{,}25; 0{,}75)$.

Pierwsze dwa przypadki już omówiliśmy. W pierwszym z nich wiemy, że następnym bitem zakodowanej wiadomości będzie 0. Aby „rozciągnąć” przedział, musimy wypisać ten bit do bufora, a następnie przesunąć bity L oraz H w lewo. Dzięki temu, bit ten „wypadnie”, a my zostaniemy z szerszym przedziałem. Operację przesunięcia bitów osiągamy pomnożeniem liczb przez 2. W drugim przypadku wiemy, że następnym bitem wiadomości będzie 1. Aby jednak po przesunięciu otrzymać przedział zawarty w $[0; 1)$, musimy najpierw odjąć 0,5. Jest to równoznaczne ze zmianą pierwszego bitu po przecinku z 1 na 0. Otrzymujemy więc następujące funkcje, które będą wykorzystywane odpowiednio w pierwszym i drugim przypadku:

$$E_1 : [0; 0{,}5) \to [0; 1), \qquad E_1(x) = 2x; \tag{7}$$

$$E_2 : [0{,}5; 1) \to [0; 1), \qquad E_2(x) = 2(x - 0.5). \tag{8}$$

Trzeci przypadek jest najbardziej skomplikowany. Wartość pierwszego bitu po przecinku L (0) jest różna od wartości pierwszego bitu po przecinku H (1). Nie możemy więc jednoznacznie wypisać go do bufora, ponieważ nie wiemy jeszcze, czy jest to 0, czy 1. Nie możemy też jednak pozostać bezczynni, ponieważ przedział ten może stać się nieskończenie mały w obrębie $[0{,}499\ldots, 0{,}500\ldots)$. Pierwszym krokiem w rozwiązaniu tego problemu jest zauważenie, że finalna liczba kodu znajdzie się albo w $[0{,}25; 0{,}5)$, albo w $[0{,}5; 0{,}75)$. W pierwszym przypadku bitem do wypisania będzie 0, a w drugim przypadku będzie to 1. Nie wiemy jednak jeszcze, która z tych sytuacji okaże się prawdą. Wstrzymujemy się więc z wypisaniem bitu i definiujemy następującą funkcję skalującą:

$$E_3 : [0{,}25; 0{,}75) \to [0; 1), \qquad E_3(x) = 2(x - 0.25). \tag{9}$$

Warto zauważyć, że przedział $[0{,}25; 0{,}5)$ po mapowaniu staje się przedziałem $[0; 0{,}5)$. Analogicznie, przedział $[0{,}5; 0{,}75)$ staje się przedziałem $[0{,}5; 1{,}0)$. Jeżeli więc po wybraniu podprzedziału następnego kodowanego znaku znajdziemy się w dolnej połowie przedziału (przypadek E_1), to będziemy wiedzieć, że przed skalowaniem funkcją E_3 powinniśmy „wejść” w $[0{,}25; 0{,}5)$. Zgodnie z wcześniejszą analizą oznacza to, że pierwszym bitem do wypisania jest 0. Podobnie, jeżeli podprzedział następnego symbolu trafi w górną połowę przedziału (przypadek E_2), to będziemy wiedzieć, że przed skalowaniem E_3 znaleźliśmy się w $[0{,}5; 0{,}75)$, więc poprawnym bitem do wypisania był 1. Załóżmy, że trafiliśmy w $[0{,}25; 0{,}5)$ przed skalowaniem. Nie możemy wypisać tylko bitu 0, ponieważ byłoby to równoznaczne z podprzedziałem $[0; 0{,}5)$. Zauważmy jednak, że część ułamkowa każdej liczby z przedziału $[0{,}25; 0{,}5)$ rozpoczyna się bitami

01. Wypisujemy więc te bity do kodu wiadomości. Jeżeli natomiast podprzedziałem okazałby się $[0{,}5; 0{,}75)$ — wypisalibyśmy 10.

Co jednak w sytuacji, gdy trafiamy do przypadku E_3 więcej niż jeden raz? Załóżmy, że wybieramy podprzedział $[0{,}4; 0{,}6)$. Po pierwszym skalowaniu dostaniemy $[0{,}3; 0{,}7)$. Przedział ten nadal mieści się w $[0{,}25; 0{,}75)$, więc skalujemy jeszcze raz i dostajemy $[0{,}1; 0{,}9)$. W tym momencie dzielimy przedział proporcjonalnie do prawdopodobieństw i wybieramy podprzedział następnego kodowanego symbolu. Załóżmy, że wybranym podprzedziałem jest $[0{,}1; 0{,}4)$. Mieści się on w dolnej połowie, co zgodnie z wyjaśnieniem z poprzedniego akapitu oznacza, że przed skalowaniem trafilibyśmy do $[0{,}25; 0{,}5)$. My jednak dwukrotnie skalowaliśmy przedział funkcją E_3. Przedział $[0{,}25; 0{,}5)$ to przedział po pierwszym skalowaniu:

$$\begin{aligned} E_3([L,H)) &= 2[L - 0{,}25; H - 0{,}25) \\ &= [2L - 0{,}5; 2H - 0{,}5] \\ &= [0{,}25; 0{,}5). \end{aligned} \tag{10}$$

My musimy „wrócić” o jeszcze jeden poziom wyżej. Dostajemy więc granice przedziału przed pierwszym mapowaniem E_3:

$$\begin{aligned} &2L - 0{,}5 = 0{,}25 \\ &2L = 0{,}75 \\ &L = 0{,}375, \end{aligned} \tag{11}$$

$$\begin{aligned} &2H - 0{,}5 = 0{,}5 \\ &2H = 1 \\ &H = 0{,}5. \end{aligned} \tag{12}$$

Przypominam, że zaczynaliśmy z przedziałem $[0{,}4; 0{,}6)$. Po dwóch skalowaniach E_3 ustaliliśmy, że finalna liczba będzie zawierać się w $[0{,}375; 0{,}5)$, a więc w $[0{,}4; 0{,}5)$. Zauważmy, że binarna reprezentacja części ułamkowej każdej liczby w tym przedziale zaczyna się od 011. Wypisujemy więc te bity do bufora wyjściowego. Jeżeli skalowanie E_3 byłoby konieczne trzy razy, to finalnym podprzedziałem okazałby się $[0{,}4375; 0{,}5)$, w którym część ułamkowa każdej liczby zaczyna się od bitów 0111. Warto w tym momencie zauważyć strukturę tych liczb. Jeżeli przypadek E_3 pojawi się n razy z rzędu, to wypisujemy bit narzucony przez ostatni wybrany podprzedział (sytuacja E_1 lub E_2), a następnie wypisujemy n bitów przeciwnych. Formalny dowód tego twierdzenia znajduje się w dodatku A.

Wprowadzamy więc nową zmienną stanu: *licznik*. Zmienna ta przechowywać będzie informację o tym, ile razy dokonaliśmy mapowania E_3. Przy każdym wypisaniu bitów do bufora, będziemy również wypisywać *licznik* bitów przeciwnych oraz zerować *licznik*. Definiujemy więc formalnie następujące zasady renormalizacji, które będą wykonywane w pętli:

1. $[L; H) \subseteq [0; 0{,}5)$: Wypisz 0, a następnie wypisz 1 *licznik* razy. Ustaw $licznik = 0$. Rozszerz przedział funkcją E_1.

2. $[L; H) \subseteq [0{,}5; 1{,}0)$: Wypisz 1, a następnie wypisz 0 *licznik* razy. Ustaw $licznik = 0$. Rozszerz przedział funkcją E_2.

3. $[L;H) \subseteq [0{,}25;0{,}75)$: Zwiększ licznik: $licznik = licznik + 1$. Rozszerz przedział funkcją E_3.

Uwaga 1. Warto zauważyć, że powyższe warunki nie mają zastosowania dla każdego przedziału. Przykładem jest przedział $[0{,}1;0{,}9)$. Kluczem jest tu zrozumienie, że renormalizacja i kodowanie znaku to dwa osobne procesy. Celem renormalizacji jest jedynie rozszerzenie przedziału potrzebne do uniknięcia problemów związanych z dokładnością liczb. Osiągamy to przez wypisanie bitów, które są już pewne (powyższe przypadki 1-3). „Kodowanie” nowego znaku ma miejsce tylko wtedy, gdy przedział nie wymaga renormalizacji, np. w przypadku $[0{,}1;0{,}9)$. Wtedy dzielimy przedział na podprzedziały i wybieramy podprzedział odpowiadający następnemu kodowanemu symbolowi.

3.3.1. Sekwencja końcowa

Ostatnim kodowanym symbolem jest # (EOF). Po zakończeniu pętli renormalizacyjnej otrzymamy przedział $[L;H)$, który spełnia przynajmniej jeden z warunków:

$$L < 0{,}25 < 0{,}5 \leq H \tag{13}$$

lub

$$L < 0{,}5 < 0{,}75 \leq H. \tag{14}$$

Na koniec musimy wybrać jakąś liczbę z tego przedziału, która jednoznacznie doprowadzi dekoder do podprzedziału #. Większość praktycznych implementacji wykorzystuje następujące zasady [4]:

1. Jeśli $L < 0{,}25$, wyślij 0, a następnie wyślij 1 $(licznik + 1)$ razy.

2. W przeciwnym razie, wyślij 1, a następnie wyślij 0 $(licznik + 1)$ razy.

Pomyślmy, co te warunki oznaczają. Pierwszy z nich jest podobny do sytuacji, w której wpadamy w przypadek E_1. Wysyłamy 0 oraz 1 *licznik* razy. W tym wypadku wysyłamy dodatkowo jeszcze jedno 1. Jest to konieczne, aby zawęzić ostateczny przedział z $[0;0{,}5)$ do $[0{,}25;0{,}5)$. Zobaczmy co by się stało, gdybyśmy tego nie zrobili. Załóżmy, że przedziałem # jest $[0{,}3;0{,}7)$. Mapowanie E_3 zwiększa licznik do 1 i skaluje przedział do $[0{,}1;0{,}9)$. Jeśli w tym momencie wysłalibyśmy jedynie bity 01, ich ułamkowa wartość dziesiętna wyniosłaby 0,25. Zauważmy, że liczba ta leży poza przedziałem $[0{,}3;0{,}7)$, co doprowadziłoby dekoder do błędnego symbolu. Musimy więc wysłać jeszcze jeden dodatkowy bit o wartości 1, aby otrzymać liczbę 0,375.

Drugi warunek jest podobny do sytuacji E_2. Symetrycznie z warunkiem poprzednim, tutaj również wysyłamy $(licznik + 1)$ bitów przeciwnych. W tym wypadku wizualizacja jest trochę prostsza, ponieważ wysyłana liczba zawsze jest równa 0,5. Zgodnie z warunkiem drugim, 0,5 musi być częścią ostatniego przedziału, dzięki czemu dekoder jednoznacznie odkoduje znak #.

3.4. Dekodowanie wiadomości

Dekodowanie polega na wykonywaniu tych samych kroków, które wykonywał koder. Różnica polega jedynie na tym, że zamiast generować bity, odczytujemy je z gotowego strumienia. Dekoder postępuje zgodnie z następującym algorytmem:

1. Ustaw $L = 0$, $H = 1$.

2. Wczytaj pierwsze n bitów wiadomości do zmiennej *value*.

3. Wykonuj aż do odkodowania EOF:

   3.1. Znajdź symbol, w którego podprzedziale znajduje się *value*.

   3.2. Wypisz ten symbol.

   3.3. Zaktualizuj L oraz H, zawężając je do granic znalezionego podprzedziału.

   3.4. Wykonuj pętlę renormalizacyjną dla L, H oraz *value* tak długo, jak przedział tego wymaga. Po każdej renormalizacji wczytaj następny bit wiadomości do *value*.

Warto zauważyć, że aktualny kod wiadomości (*value*) jest skalowany wraz z przedziałem. Jeżeli koder w tym momencie przesunął bity w lewo, to dekoder musi zrobić to samo, aby ich działania były zsynchronizowane. Kolejnym ważnym szczegółem jest fakt, że operujemy tylko na n bitach kodu. Jest to konieczne, ponieważ komputer może nie być w stanie utrzymać całej wiadomości w pamięci. Zamiast tego, po każdej renormalizacji (przesunięciu bitów w lewo) wczytujemy następny bit ze strumienia. Dokładniejszy opis działania prawdziwych implementacji znajduje się w sekcji 5.

4. Przykłady obliczeń

Na potrzeby poniższych przykładów zakładamy, że komputer jest w stanie utrzymać pełne wartości liczb w pamięci.

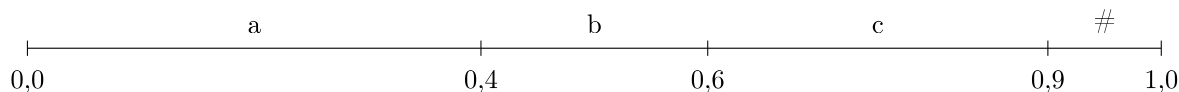
4.1. Przykład kodowania

Aby w pełni zrozumieć działanie algorytmu wraz z mechanizmem renormalizacji i sekwencją końcową, prześledźmy proces kodowania krótkiej wiadomości. Zakładamy alfabet składający się z czterech symboli $\{a, b, c, \#\}$, dla których określono następujące prawdopodobieństwa wystąpienia: $P(a) = 0{,}4$, $P(b) = 0{,}2$, $P(c) = 0{,}3$ oraz $P(\#) = 0{,}1$. Symbol # oznacza koniec pliku (EOF).

Naszym celem jest zakodowanie wiadomości „ba#”. Stan początkowy kodera to przedział $[0{,}0; 1{,}0)$ oraz zmienna $licznik = 0$.

Krok 1: Kodowanie symbolu b

Na początku mapujemy symbole na bazowy przedział $[0{,}0; 1{,}0)$ zgodnie z ich prawdopodobieństwami:



Wybieramy podprzedział odpowiadający symbolowi b, czyli $[0{,}4; 0{,}6)$. Następnie sprawdzamy warunki renormalizacji:

1. Przedział $[0{,}4; 0{,}6)$ mieści się w całości w środkowej połowie $[0{,}25; 0{,}75)$, co oznacza przypadek E_3.

   - Skalujemy: $L = 2(0{,}4 - 0{,}25) = 0{,}3$; $H = 2(0{,}6 - 0{,}25) = 0{,}7$.

   - Inkrementujemy licznik: $licznik = 1$.

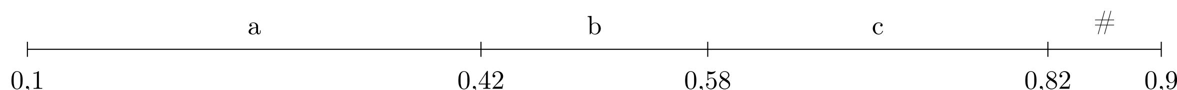
2. Nowy przedział [0,3; 0,7) ponownie spełnia warunek E_3.

   - Skalujemy: $L = 2(0{,}3 - 0{,}25) = 0{,}1$; $H = 2(0{,}7 - 0{,}25) = 0{,}9$.
   - Inkrementujemy licznik: $licznik = 2$.

Przedział [0,1; 0,9) nie wymaga renormalizacji. Zakończyliśmy przetwarzanie pierwszego znaku. Bufor wyjściowy jest nadal pusty.

Krok 2: Kodowanie symbolu *a*

Dzielimy aktualny przedział [0,1; 0,9) (o szerokości 0,8) na nowe podprzedziały:



Wybieramy podprzedział odpowiadający symbolowi a, czyli [0,1; 0,42). Sprawdzamy renormalizację:

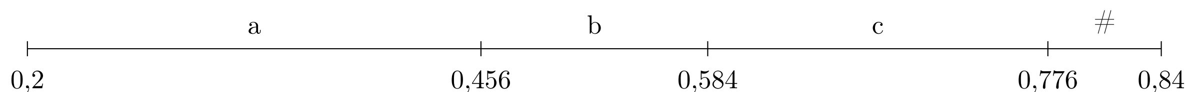
1. Przedział [0,1; 0,42) zawiera się w dolnej połowie [0; 0,5), co oznacza przypadek E_1.

   - Wypisujemy bity: Algorytm nakazuje wypisać 0, a następnie *licznik* jedynek. $licznik = 2$, więc wypisujemy 011.
   - Zerujemy licznik: $licznik = 0$.
   - Skalujemy: $L = 2(0{,}1) = 0{,}2$; $H = 2(0{,}42) = 0{,}84$.

Przedział [0,2; 0,84) nie wymaga renormalizacji. Zawartość bufora to `011`.

Krok 3: Kodowanie symbolu #

Dzielimy przedział [0,2; 0,84) (o szerokości 0,64) na nowe podprzedziały:



Wybieramy podprzedział odpowiadający symbolowi #, czyli [0,776; 0,84). Sprawdzamy renormalizację:

1. Przedział [0,776; 0,84) leży w górnej połowie [0,5; 1,0) (przypadek E_2).

   - Wypisujemy bit: `1` (licznik to 0, więc nie wypisujemy zer). Bufor: `0111`.
   - Skalujemy: $L = 2(0{,}776 - 0{,}5) = 0{,}552$; $H = 2(0{,}84 - 0{,}5) = 0{,}68$.

2. Nowy przedział [0,552; 0,68) nadal leży w E_2.

   - Wypisujemy bit: `1`. Bufor: `01111`.
   - Skalujemy: $L = 2(0{,}552 - 0{,}5) = 0{,}104$; $H = 2(0{,}68 - 0{,}5) = 0{,}36$.

3. Nowy przedział [0,104; 0,36) leży w dolnej połowie, co oznacza przypadek E_1.

- Wypisujemy bit: `0`. Bufor: `011110`.
- Skalujemy: $L = 2(0{,}104) = 0{,}208$; $H = 2(0{,}36) = 0{,}72$.

Przedział $[0{,}208; 0{,}72)$ nie wymaga renormalizacji. Zawartość bufora to `011110`.

Krok 4: Sekwencja końcowa

Analizujemy ostateczny przedział $[0{,}208; 0{,}72)$. Ponieważ dolna granica $L = 0{,}208$ jest mniejsza od 0,25, stosujemy pierwszy warunek sekwencji końcowej. Wypisujemy więc bit 0, a następnie $(licznik + 1)$ jedynek, gdzie $licznik = 0$. Daje to ciąg `01`, który dodajemy do bufora.

Ostateczny kod binarny dla wiadomości „ba#” to **01111001**.

4.2. Przykład dekodowania

Otrzymujemy źródło z alfabetem $\{a,b,c,\#\}$ oraz prawdopodobieństwami wystąpienia $P(a) = 0{,}4$, $P(b) = 0{,}2$, $P(c) = 0{,}3$ oraz $P(\#) = 0{,}1$. Otrzymujemy również zakodowaną wiadomość 01111001. Naszym celem jest ją odkodować.

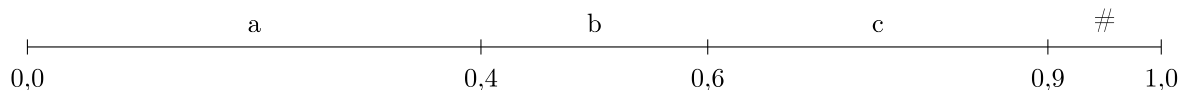
Na początku wczytujemy kod wiadomości i interpretujemy go jako ułamek binarny V:

$$V = 0{,}01111001_2 = \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} = 0{,}47265625. \tag{15}$$

Początkowy przedział to $L = 0{,}0$ oraz $H = 1{,}0$.

Krok 1: Odkodowanie pierwszego symbolu

Dzielimy przedział $[0{,}0; 1{,}0)$ na podprzedziały:



Porównujemy naszą wartość $V \approx 0{,}4726$ z granicami podprzedziałów. Wartość ta mieści się w zakresie $[0{,}4; 0{,}6)$. Pierwszym znakiem wiadomości jest więc b. Aktualizujemy przedział do $[0{,}4; 0{,}6)$.

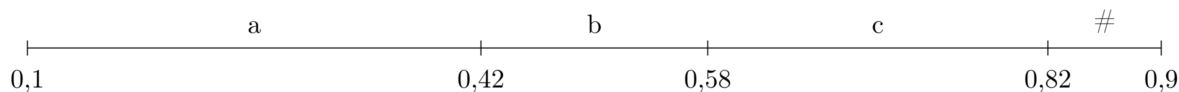
Sprawdzamy warunki renormalizacji (pamiętając, że wartość V musi być przekształcana synchronicznie z granicami L i H):

1. Przedział $[0{,}4; 0{,}6)$ leży w środku zakresu, co oznacza przypadek E_3.
   - Skalujemy granice: $L = 2(0{,}4 - 0{,}25) = 0{,}3$; $H = 2(0{,}6 - 0{,}25) = 0{,}7$.
   - Skalujemy V: $V = 2(0{,}47265625 - 0{,}25) = 0{,}4453125$.
2. Nowy przedział $[0{,}3; 0{,}7)$ ponownie oznacza przypadek E_3.
   - Skalujemy granice: $L = 2(0{,}3 - 0{,}25) = 0{,}1$; $H = 2(0{,}7 - 0{,}25) = 0{,}9$.
   - Skalujemy V: $V = 2(0{,}4453125 - 0{,}25) = 0{,}390625$.

Przedział $[0{,}1; 0{,}9)$ nie wymaga renormalizacji. Aktualna wartość kodu to $V = 0{,}390625$. Aktualna odkodowana wiadomość to „b”.

Krok 2: Odkodowanie drugiego symbolu

Dzielimy aktualny przedział $[0{,}1; 0{,}9)$ (o szerokości 0,8) na podprzedziały:



$V = 0{,}390625$ wpada w zakres $[0{,}1; 0{,}42)$. Drugim symbolem jest więc a. Aktualizujemy przedział do $[0{,}1; 0{,}42)$.

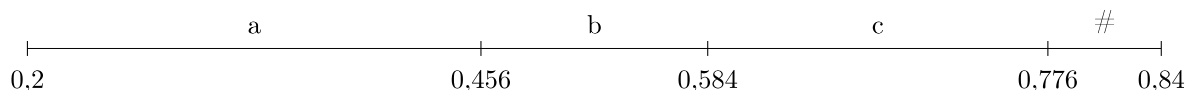
Sprawdzamy warunki renormalizacji:

1. Przedział $[0{,}1; 0{,}42)$ leży w całości w dolnej połowie, co oznacza przypadek E_1.
   - Skalujemy granice: $L = 2(0{,}1) = 0{,}2$; $H = 2(0{,}42) = 0{,}84$.
   - Skalujemy V: $V = 2(0{,}390625) = 0{,}78125$.

Przedział $[0{,}2; 0{,}84)$ nie wymaga renormalizacji. Aktualna wartość kodu to $V = 0{,}78125$. Aktualna odkodowana wiadomość to „ba”.

Krok 3: Odkodowanie trzeciego symbolu

Dzielimy przedział $[0{,}2; 0{,}84)$ (o szerokości 0,64) na podprzedziały:



$V = 0{,}78125$ wpada w zakres $[0{,}776; 0{,}84)$. Trzecim symbolem jest więc #. Ponieważ jest to symbol oznaczający koniec kodu, pętla dekodera zostaje przerwana. Ostateczna odkodowana wiadomość to „ba#” (lub „ba” po usunięciu #).

Uwaga 2. Spostrzegawczy czytelnik może zauważyć, że wiadomość „ba” mogła zostać zakodowana na mniejszej liczbie bitów używając innego kodu. Rzeczywiście wygląda na to, że użycie kodowania arytmetycznego mogło wręcz powiększyć ilość bitów potrzebnych do zapisania wiadomości. Problem ten wynika z faktu, że wykorzystany rozkład prawdopodobieństwa nijak ma się do częstości występowania znaków w wiadomości „ba#”. Pokazuje to, że nawet ważniejsza od samej metody kodowania jest poprawna reprezentacja źródła dla kodowanej wiadomości.

5. Implementacja na liczbach całkowitych

Jednym z najważniejszych artykułów naukowych w historii kodowania arytmetycznego jest „Arithmetic coding for data compression” [4], opublikowany w 1987 roku przez Iana H. Wittena, Radforda M. Neala i Johna G. Cleary’ego. Jest to pierwszy artykuł, który pokazał niezawodną i wydajną implementację kodowania arytmetycznego, zawieracjący kompletny kod źródłowy kodera oraz dekodera napisany w języku C. Implementacja ta szybko stała się standardem kodowania arytmetycznego. Jako ciekawostkę warto dodać, że pierwsze prace naukowe opisujące kod arytmetyczny zostały opublikowane w 1976 roku — aż 11 lat przed omawianym artykułem.

Zamiast przechowywać wartości jako liczby zmiennoprzecinkowe, autorzy użyli liczb całkowitych. Liczba bitów określająca precyzję jest ustalona odgórnie dla wartości przedziałów i kodu (oryginalnie było

to 16 bitów). Początkowym L jest liczba 0, a początkowym H jest liczba, której binarna reprezentacja składa się z samych jedynek w wybranej precyzji (dla 16 bitów jest to $2^{16}-1$). Odpowiednikiem matematycznej wartości 0,5 jest w tym układzie dokładnie połowa zakresu (czyli 2^{15} dla precyzji 16-bitowej). Ważnym szczegółem jest fakt, że zarówno koder jak i dekoder używają tej samej precyzji. Wykorzystanie liczb całkowitych jest wydajniejsze, dokładniejsze oraz mniej skomplikowane, niż korzystanie z liczb zmiennoprzecinkowych. Korzyści z tego podejścia są widoczne szczególnie przy rozważaniu renormalizacji, gdzie „mnożenie przez 2" jest równoznaczne z prostym przesunięciem bitów w lewo. Dokładny opis implementacji znajduje się w oryginalnym artykule, do którego przeczytania zachęcani są zainteresowani czytelnicy.

6. Podsumowanie

W niniejszym artykule omówiono teoretyczny sposób działania kodera oraz dekodera arytmetycznego. Szczególna uwaga poświęcona została intuicyjnemu wyjaśnieniu celu i mechanizmu renormalizacji. Mechanizm ten jest jednym z najważniejszych detali kodowania arytmetycznego, ale cechuje się on znacznie wyższym poziomem złożoności niż reszta algorytmu. Jest on również często słabo wyjaśniany przez autorów podobnych publikacji. Przydatnym narzędziem do głębszego zrozumienia kodowania arytmetycznego są przykłady kodowania i dekodowania zawarte w sekcji 4., obliczone krok po kroku z przystępnymi ilustracjami. Poruszony został również temat praktycznej implementacji kodowania arytmetycznego, która różni się od matematycznej teorii wykorzystaniem liczb całkowitych do symulowania przedziału.

A. Dowód poprawności E_3

Lemat 1. *Niech $E_3^n(x)$ oznacza rekurencyjne wykonanie funkcji E_3 n razy:* $E_3^n = \underbrace{E_3 \circ E_3 \circ \cdots \circ E_3}_{n \text{ razy}}$.

Wtedy:

$$E_3^n(x) = 2^n x - \frac{1}{2}(2^n - 1). \tag{16}$$

Dowód. Użyjemy indukcji na n.

Baza: niech $n_0 = 1$. Wtedy $E_3^1(x) = 2^1 x - \frac{1}{2}(2^1 - 1) = 2x - 0{,}5 = 2(x - 0{,}25)$.

Załóżmy, że $E_3^n(x) = 2^n x - \frac{1}{2}(2^n - 1)$. Wtedy:

$$\begin{aligned} E_3^{n+1}(x) &= E_3(E_3^n(x)) \\ &= 2(E_3^n(x)) - 0{,}5 \qquad\qquad (\text{z definicji } E_3) \\ &= 2(2^n x - \frac{1}{2}(2^n - 1)) - \frac{1}{2} \\ &= 2^{n+1}x - (2^n - 1) - \frac{1}{2} \\ &= 2^{n+1}x - 2^n + \frac{1}{2} \\ &= 2^{n+1}x - \frac{1}{2}(2^{n+1} - 1). \end{aligned} \tag{17}$$

□

E_3^n pozwala nam na obliczenie granic przedziału po wpadnięciu w przypadek E_3 n razy z rzędu. W dalszej części dowodu przyda nam się również odwrotność E_3^n:

$$\begin{aligned}
2^n x - \frac{1}{2}(2^n - 1) &= y \\
2^n x &= y + \frac{1}{2}(2^n - 1) \\
x &= \frac{y + \frac{1}{2}(2^n - 1)}{2^n} \\
(E_3^n)^{-1}(y) &= \frac{y + \frac{1}{2}(2^n - 1)}{2^n}.
\end{aligned} \tag{18}$$

Zanim przejdziemy dalej, pomyślmy o strukturze tego dowodu. Intuicyjnie rozumiemy już, dlaczego przypadek E_3 działa. Zaczynamy z pewnym przedziałem będącym w obrębie środka, na przykład $[0{,}3; 0{,}7)$. Następnie skalujemy go funkcją E_3 n razy (tutaj 1 raz), aż nie będzie wymagał on renormalizacji. W tym wypadku otrzymujemy przedział $[0{,}1; 0{,}9)$. Po skalowaniu wybieramy podprzedział następnego kodowanego symbolu. Jeżeli podprzedział ten znajdzie się w $[0; 0{,}5)$, to „wrócenie" w górę pokaże nam, że powinniśmy wejść również w dolną połowę przedziału $[0{,}3; 0{,}7)$. Dla przykładu załóżmy, że wybranemu symbolowi odpowiada podprzedział $[0{,}2; 0{,}4)$. Aby „wrócić" do przedziału $[0{,}3; 0{,}7)$, musimy wykonać odwrotność skalowania E_3 dla podprzedzialu $[0{,}2; 0{,}4)$:

$$[L; H) = (E_3^1)^{-1}([0{,}2; 0{,}4)) = [\frac{0{,}2}{2} + 0{,}25; \frac{0{,}4}{2} + 0{,}25) = [0{,}35; 0{,}45). \tag{19}$$

Wiemy teraz, że kod naszej wiadomości w odniesieniu do przedziału przed pierwszym skalowaniem E_3 znajduje się w $[0{,}35; 0{,}45)$. Kluczowe jest zrozumienie, co kryje się pod frazą „w odniesieniu do przedziału". Zaczynaliśmy z przedziałem $[0{,}3; 0{,}7)$. Następnie zrobiliśmy renormalizację E_3 i wybraliśmy symbol $[0{,}2; 0{,}4)$. Podprzedział $[0{,}2; 0{,}4)$ jest podprzedziałem w kontekście po skalowaniu E_3. My jednak wróciliśmy w górę odwrotnością skalowania, aby zmienić kontekst z „kod wiadomości znajduje się w podprzedziale $[0{,}2; 0{,}4)$ przedziału $[0{,}1; 0{,}9)$" na „kod wiadomości znajduje się w podprzedziale $[0{,}35; 0{,}45)$ przedziału $[0{,}3; 0{,}7)$". Twierdzenia te są sobie równoznaczne. Różnią się jedynie tym, że pierwsze z nich odnosi się do przedziału po skalowaniu, a drugie do przedziału przed skalowaniem. Zrozumienie pojęcia „odnoszenia się do przedziału" jest konieczne do zrozumienia następującego dowodu.

Równania wyprowadzone i udowodnione w lemacie 1 oraz równaniu 18 pozwalają nam na obliczanie przedziału i jego odwrotności po n skalowaniach funkcją E_3. Zauważmy jednak, że są one prawidłowe jedynie dla ciągłych wykonań funkcji E_3. Co jednak w sytuacji, gdy po przeskalowaniu przedziału $[0{,}3; 0{,}7)$ na przedział $[0{,}1; 0{,}9)$, podprzedziałem następnego kodowanego symbolu będzie $[0{,}4; 0{,}6)$? Ten podprzedział ponownie wpadnie w przypadek E_3, ale samo zwiększenie liczby n nie pozwoli nam na obliczenie poprawnego przedziału po kolejnym skalowaniu. Zamiast tego, musimy najpierw obliczyć $E_3^1([0{,}3; 0{,}7))$, a następnie $E_3^2([0{,}4; 0{,}6))$. Równanie E_3^n jest poprawne tylko dla n skalowań jednego przedziału z rzędu, bez kodowania nowego symbolu w trakcie. Mimo to, w dowodzie musimy pokazać, że poprawną wartością kodu jest 0 oraz 1 n-razy, niezależnie od tego, ile znaków zostanie zakodowanych pomiędzy pierwszym mapowaniem E_3 a pierwszym przypadkiem E_1. Na szczęście, jak za chwilę się wyjaśni, charakterystyka funkcji E_3^n nie jest wielkim problemem.

Aby ograniczyć następujący dowód do rozsądnej długości, udowodnimy jedynie poprawność sytuacji, w której pierwszym napotkanym przypadkiem niebędącym E_3 jest przypadek E_1. Naszym celem jest więc wykazanie, że niezależnie od ilości kodowanych symboli, jeżeli pierwszym przypadkiem niebędącym E_3 jest E_1, to kod wiadomości w odniesieniu do przedziału przed pierwszym E_3 musi zaczynać się od bitu 0, po którym bit 1 następuje n razy.

Lemat 2. *Załóżmy, że pierwszym przypadkiem renormalizacji niebędącym E_3 jest E_1. Niech n będzie łączną liczbą wykonań mapowania E_3 przed wystąpieniem tego przypadku E_1. Niech $[L; H)$ będzie poprawnym podprzedziałem kodu w odniesieniu do przedziału przed pierwszym skalowaniem E_3. Wtedy:*

$$\frac{1}{2} - \frac{1}{2^{n+1}} \leq L < H < \frac{1}{2}. \tag{20}$$

Dowód. Z definicji przypadku E_1 wiemy, że przedział w momencie jego napotkania znajduje się w całości w dolnej połowie, tj. jest podzbiorem przedziału $[0; \frac{1}{2})$.

Zauważmy, że proces kodowania symboli pomiędzy mapowaniami E_3 polega wyłącznie na wyborze podprzedziału, co jedynie zawęża przestrzeń rozwiązań. Oznacza to, że każdy krok kodowania zachowuje relację zawierania. Wobec tego, aby znaleźć ostateczne granice w odniesieniu do początkowego przedziału, wystarczy cofnąć cały zbiór docelowy $[0; \frac{1}{2})$ przez n mapowań E_3.

Ponieważ funkcja $E_3(x)$ oraz jej odwrotność $(E_3^n)^{-1}(x)$ są funkcjami afinicznymi o dodatnich współczynnikach kierunkowych (są więc ciągłe i ściśle rosnące), przeciwobraz przedziału $[0; \frac{1}{2})$ jest również przedziałem ciągłym, a jego granice możemy wyznaczyć, podstawiając wartości skrajne do wzoru na odwrotność (Równanie 18):

$$(E_3^n)^{-1}(0) = \frac{0 + \frac{1}{2}(2^n - 1)}{2^n} = \frac{2^{n-1} - \frac{1}{2}}{2^n} = \frac{1}{2} - \frac{1}{2^{n+1}},$$

$$\lim_{y \to \frac{1}{2}} (E_3^n)^{-1}(y) = \frac{\frac{1}{2} + \frac{1}{2}(2^n - 1)}{2^n} = \frac{\frac{1}{2} + 2^{n-1} - \frac{1}{2}}{2^n} = \frac{2^{n-1}}{2^n} = \frac{1}{2}.$$

Otrzymujemy zatem przeciwobraz równy $[\frac{1}{2} - \frac{1}{2^{n+1}}; \frac{1}{2})$. Ponieważ ostateczny podprzedział przed przypadkiem E_1 zawiera się w $[0; \frac{1}{2})$, to odpowiadający mu podprzedział początkowy $[L; H)$ musi zawierać się w jego przeciwobrazie:

$$[L; H) \subseteq \left[\frac{1}{2} - \frac{1}{2^{n+1}}; \frac{1}{2}\right), \tag{21}$$

co dowodzi, że $\frac{1}{2} - \frac{1}{2^{n+1}} \leq L < H < \frac{1}{2}$. □

Twierdzenie 1. *Jeżeli pierwszym napotkanym przypadkiem renormalizacyjnym niebędącym E_3 jest E_1, to binarny kod wiadomości w odniesieniu do przedziału przed pierwszym skalowaniem E_3 rozpoczyna się bitami 01^n, gdzie 1^n oznacza wypisanie bitu 1 tyle razy, ile wykonane zostało skalowanie E_3.*

Dowód. Niech $[L; H)$ będzie prawidłowym podprzedziałem kodu w odniesieniu do przedziału przed pierwszym skalowaniem E_3. Wtedy zgodnie z Lematem 2: $\frac{1}{2} - \frac{1}{2^{n+1}} \leq L < H < \frac{1}{2}$. Szczególnie ważna jest dla nas nierówność $\frac{1}{2} - \frac{1}{2^{n+1}} \leq L$. Zauważmy, że dolną granicę $\frac{1}{2} - \frac{1}{2^{n+1}}$ możemy rozwinąć do sumy potęg

dwójki (zakładamy skończony zapis binarny liczby $\frac{1}{2}$):

$$\frac{1}{2} - \frac{1}{2^{n+1}} = \sum_{i=2}^{n+1} 2^{-i}. \tag{22}$$

W systemie binarnym wartość ta odpowiada dokładnie ułamkowi $0{,}0\underbrace{1\ldots1}_{n \text{ razy}}{}_2$, gdzie bity na ułamkowych pozycjach od 2 do $n+1$ włącznie są równe 1. Ponieważ górna granica $H < \frac{1}{2}$, wszystkie liczby w przedziale $[L;H)$ muszą mieć bit 0 na pierwszej pozycji ułamkowej. Jednocześnie, ponieważ są one większe lub równe dolnej granicy L, ich kolejne n bitów musi być równe 1. Dowodzi to, że kod binarny każdej wiadomości z przedziału $[L;H)$ w odniesieniu do przedziału przed pierwszym skalowaniem E_3 rozpoczyna się bitami 01^n. □

Dowód poprawności renormalizacji E_3 kończącej się przypadkiem E_2 postępuje analogicznie i pozostawiony został jako ćwiczenie dla zainteresowanych czytelników.

Literatura

1. D.A. Huffman, *A Method for the Construction of Minimum-Redundancy Codes*, Proceedings of the IRE 40 (1952), pp. 1098-1101.
2. K. Sayood, *Introduction to Data Compression 3rd edition*, Morgan Kaufmann Publishers, San Francisco 2006.
3. C.E. Shannon, *A Mathematical Theory of Communication*, Bell Syst. Tech. J. 27 (1948), pp. 379-423, 623-656.
4. I.H. Witten, R.M. Neal, J.G. Cleary, *Arithmetic coding for data compression*, Communications of the ACM 30 (1987), pp. 520-540.