

Spis treści

1	Wstęp	1
1.1	"Prawo" Kompresji bezstratnej	1
2	Kodowania	2
2.1	Modelowanie danych	2
2.2	Średnia długość kodu	2
2.3	Jednoznaczna dekodowalność	2
2.3.1	Nierówność Krafta	2
2.3.2	Kod prefiksowy	2
2.4	Kod natychmiastowy	2
2.5	Statyczny Kod Huffmana	2
2.6	Kodowanie Shannon-Fano	3
2.7	Kodowanie Tunstalla	3
2.8	Kodowanie Golomba	3
2.9	Dynamiczne kodowanie Huffmana	3
3	Teoria informacji	4
3.1	Miara informacji	4
3.2	Entropia	4
3.2.1	Entropia źródła	4
3.2.2	Entropia Pierwszego Rzędu	4
4	Kodowanie uniwersalne	4
4.1	Kodowanie Eliasza	4
4.1.1	γ	4
4.1.2	δ	5
4.1.3	ω	5
4.2	Kodowanie Fibonacciego	5
5	Kodowanie arytmetyczne	5
5.1	Algorytm	5
5.2	Wizualizacja	5
6	Kodowanie Słownikowe	5
6.1	Statyczne kodowanie słownikowe	5
6.2	LZ77	6
6.3	LZ78	6
6.4	LZW	6
7	bzip2	6

1 Wstęp

Wyróżniamy dwa rodzaje kompresji. W kompresji stratnej dopuszczalny jest pewien stopień straty informacji wejściowej. W kompresji bezstratnej nie jest to dopuszczalne.

1.1 "Prawo" Kompresji bezstratnej

Nie istnieje algorytm, który potrafi zmniejszyć rozmiar dowolnych danych

- Kompresja bezstratna musi być bijekcją
- Dowolne dane przyjmują postać ciągu bitów długości n . Jest 2^n takich ciągów.
- Danych krótszych niż n , np.: o jeden jest 2^{n-1}
- Nie da się stworzyć bijekcji z zbioru o mocy 2^n do zbioru o mocy 2^{n-1}

Wniosek jest taki, że koniecznym jest konstruowanie kompresji bezstratnej na podzbiorach danych, takich jak np.: obrazów, dźwięków, tekstów.

2 Kodowania

Kodowanie to przyporządkowanie elementom jakiegoś alfabetu ciągu binarnych. Przykładami kodowania są: ASCII, UTF-8 oraz inne. Typowym jest konstruowanie kodowania pod konkretny zestaw danych, optymalizując je pod kątem częstości występowania poszczególnych elementów.

2.1 Modelowanie danych

Rozważmy ciąg: $a_n = 9, 11, 11, 11, 14, 13, 15, 17, 16, 17, 20, 21$. $\max(a_n) = 21$ stąd koniecznym jest 5 bitów na element. Ale jeśli wykorzystamy wzór $e_n = a_n - n + 8$ do stworzenia nowego ciągu, to ten ciąg przyjmuje postać: $0, 1, 0, -1, 1, -1, 0, 1, -1, -1, 1, 1$. Teraz wystarczy tylko 2 bity na zakodowanie elementu.

2.2 Średnia długość kodu

$$I = \sum_{i=1}^n p_i \cdot l_i$$

gdzie p_i to prawdopodobieństwo wystąpienia elementu i , a l_i to długość kodu dla elementu i .

2.3 Jednoznaczna dekodowalność

Jeśli dla dowolnego ciągu znaków istnieje tylko jedno jego rozkodowanie to kod jest jednoznacznie dekodowalny. Aby sprawdzić czy kod jest jednoznacznie dekodowalny, należy zastosować następujący algorytm.

1. Stwórz pustą listę
2. Dla każdej pary słów kodowych sprawdź czy jedno jest prefiksem drugiego. Jeśli tak, dodaj sufiks drugiego słowa do listy, jeśli już go tam nie ma.
3. Jeśli na liście jest słowo kodowe, to kod nie jest jednoznacznie dekodowalny.

2.3.1 Nierówność Krafta

Jeżeli \mathcal{C} jest kodem jednoznacznie dekodowalnym z n słowami to:

$$K(\mathcal{C}) = \sum_{i=1}^n 2^{-l_i} \leq 1$$

Jest to warunek konieczny bycia kodem jednoznacznie dekodowalnym.

2.3.2 Kod prefiksowy

Kod w którym żadne słowo kodowe nie jest prefiksem innego słowa kodowego. Wszystkie kody prefiksowe są jednoznacznie dekodowalne.

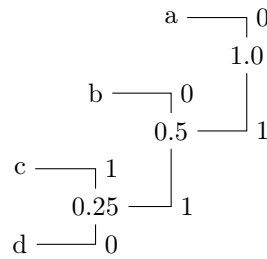
2.4 Kod natychmiastowy

Jest kodem pozwalającym stwierdzić w którym miejscu zakończone jest słowo kodowe w momencie odczytania ostatniej litery.

2.5 Statyczny Kod Huffmana

Kod Huffmana to kod prefiksowy o minimalnej średniej długości kodu. Są one optymalne wśród kodów prefiksowych.

Dla alfabetu \mathcal{A} o długości n i prawdopodobieństwach wystąpienia p_1, \dots, p_n algorytm tworzenia kodu Huffmana wygląda następująco: Znajdź dwa najrzadziej występujące elementy i połącz je w jeden element o prawdopodobieństwie $p_1 + p_2$. Rozróżnij je 0 lub 1. Powtórz ten krok na liście $n - 1$ długiej aż zostanie jeden element.

Diagram 1: Przykład kodu Huffmana dla $P(a) = 0.5, P(b) = 0.25, P(c) = 0.15, P(d) = 0.1$

2.6 Kodowanie Shannon-Fano

Dla symboli a_1, \dots, a_n o prawdopodobieństwach p_1, \dots, p_n , ustalmy kody długości $l_n = \lceil -\log p_i \rceil$. Następnie zdefiniujmy zmienne pomocnicze w_1, \dots, w_n jako:

$$w_1 = 0, w_j = \sum_{i=1}^{j-1} 2^{l_j - l_i}$$

Jeżeli $\lceil \log w_j \rceil = l_j$ to j -te słowo kodowe jest binarną reprezentacją w_j . Jeżeli $\lceil \log w_j \rceil < l_j$ to reprezentację uzupełniamy zerami z lewej strony.

Dla $P(a) = \frac{1}{3}, P(b) = \frac{1}{4}, P(c) = \frac{1}{4}, P(d) = \frac{1}{6}$ mamy:

$$l_a = 2, l_b = 2, l_c = 2, l_d = 3$$

$$w_1 = 0, w_2 = 2, w_3 = 2, w_4 = 6$$

$$\text{kod}(a) = 00, \text{kod}(b) = 01, \text{kod}(c) = 10, \text{kod}(d) = 110$$

2.7 Kodowanie Tunstalla

Chcemy stworzyć kod na n bitach dla a_1, \dots, a_m symboli o prawdopodobieństwach p_1, \dots, p_m . Tworzenie kodu Tunstalla polega na iteracyjnym wyborze ze zbioru symbolu o największym prawdopodobieństwie S i łączenie go z wszystkimi innymi symbolami tworząc symbole Sa_m , nadając im prawdopodobieństwa $P \cdot p_m$. Proces ten powtarzamy aż do uzyskania kodu o długości n .

2.8 Kodowanie Golomba

Kody Golomba są parametryzowane liczbą $m > 0$. Każda liczba n jest zapisywana za pomocą $q = \lfloor \frac{n}{m} \rfloor$ oraz $r = n - q \cdot m$ w postaci

$$(q)_1(r)_2$$

2.9 Dynamiczne kodowanie Huffmana

Głównym problemem kodowania Huffmana jest konieczność znania całego ciągu danych przed rozpoczęciem kodowania. Rozwiązaniem tego problemu jest dynamiczne kodowanie, gdzie stosujemy kodowanie Huffmana dla $k+1$ symbolu na podstawie kodowania dla k symboli. W tym celu tworzymy dynamicznie drzewo, gdzie każdy liść ma wagę równą ilości wystąpień danego symbolu. Drzewo zaczyna się od liścia z symbolem EOF o wadze 0.

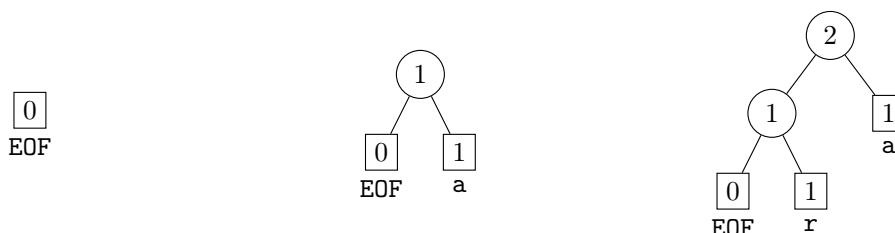


Diagram 2: Przykład kodowania dynamicznego

3 Teoria informacji

Teoria informacji to dziedzina zajmująca się przetwarzaniem informacji.

3.1 Miara informacji

Miarą informacji, którą niesie ze sobą zdarzenie A jest:

$$I(A) = -\log_x P(A)$$

gdzie x to baza systemu liczbowego. Jeśli miarą informacji jest bit to $x = 2$. Jeśli zdarzenia A i B są niezależne to:

$$I(AB) = I(A) + I(B)$$

3.2 Entropia

Entropia to miara średniej informacji przekazywanej przez źródło. Kody jednoznacznie dekodowalne w modelu z niezależnymi wystąpieniami symboli muszą mieć średnią długość co najmniej równą entropii.

3.2.1 Entropia źródła

Dla źródła danych S generującego ciąg X nad alfabetem $\mathcal{A} = \{1, 2, \dots, m\}$

$$H(S) = \lim_{n \rightarrow \infty} \frac{G_n}{n}$$

$$G_n = - \sum_i \dots \sum_j P(X_1 = i, \dots, X_n = j) \log P(X_1 = i, \dots, X_n = j)$$

3.2.2 Entropia Pierwszego Rzędu

Dla źródła informacji X , z zbiorem wiadomości (zdarzeń) A_1, \dots, A_n , gdzie $P(A_i)$ to prawdopodobieństwo wystąpienia zdarzenia A_i i zdarzenia są niezależne to entropia źródła to:

$$H(X) = \sum_{i=1}^n P(A_i) I(A_i)$$

4 Kodowanie uniwersalne

Szukamy sposobu na kodowanie dowolnej liczby $n \in \mathbb{N}$. Problem polega na skonstruowaniu kodu, który będzie jednoznacznie dekodowalny i uniwersalny. To oznacza, że ma się skalować w nieskończoność.

4.1 Kodowanie Eliasa

Kodowanie Eliasa to kodowanie uniwersalne, które wykorzystuje kodowanie unarne do zapisania długości kodu binarnego liczby n .

$$n = \lfloor \log_2(x) \rfloor + 1$$

4.1.1 γ

Jest to najprostsze z kodowań Eliasa. Polega na zakodowaniu liczby x w postaci binarnej, a następnie dodaniu przed nią liczby $n - 1$ zer.

$$\gamma(x) = 0^{n-1}(x)_2$$

$$(13)_{10} = 1101_2 \Rightarrow \gamma(13) = 0001101$$

4.1.2 δ

Cały trik kodu δ polega na zakodowaniu długości kodu binarnego liczby x przy pomocy kodu γ . Istotnym trikiem jest usunięcie najstarszego bitu z zakodowanej liczby x .

$$\delta(x) = \gamma(n) + (x)_2$$

$$(13)_{10} = 1101_2 \Rightarrow \delta(13) = 00100101$$

Jak widać, jest on bardziej efektywny dla większych liczb. Długość kodu δ to $2 \cdot \lceil \log_2(\lceil \log_2 x \rceil) \rceil - 1 + \lceil \log_2 x \rceil - 1$.

4.1.3 ω

Jest to kodowanie rekurencyjne, które działa jak kodowanie δ , ale w nieskończoność. Na koniec umieszczane jest 0, potem kodowana jest liczba $k = x$. Potem ten krok jest powtarzany dla $k = n - 1$ gdzie n to liczba bitów z poprzedniego kroku.

$$(13)_{10} = 1101_2 \Rightarrow \omega(13) = 1111010$$

4.2 Kodowanie Fibonacciego

Liczba Fibonacciego ma postać:

$$f_0 = f_1 = 1$$

$$f_n = f_{n-1} + f_{n-2} : n \geq 2$$

Kodowanie fibonacciego polega na reprezentacji liczby x jako sumę liczb fibonacciego.

$$x = \sum_{i=0} a_i \cdot f_i, a_i \in \{0, 1\}$$

$$(13)_{10} = f_7 = 1101_2 \Rightarrow Fib(13) = 0000011$$

5 Kodowanie arytmetyczne

Kodowanie arytmetyczne to kodowanie, które odwzorowuje dowolny ciąg wejściowy na liczbę z zakresu $[0, 1)$. Głównym pomysłem stojącym za algorytmem, jest iteracyjne przypisywanie coraz to mniejszych przedziałów do kolejnych symboli ciągu wejściowego.

5.1 Algorytm

Dla zakresu początkowego $[l, p) = [0, 1)$, ciągu symboli wejściowych a_j , dystrybuanaty $F(j)$ i prawdopodobieństw p_j algorytm wygląda następująco:

- $d = p - l$
- $p = l + d \cdot F(j + 1)$
- $l = l + F(j)d$

Powyższe kroki wykonujemy dla każdego symbolu ciągu wejściowego. Na koniec dostajemy zakres, z którego potem możemy wybrać dowolną liczbę jako wynik kodowania.

5.2 Wizualizacja

6 Kodowanie Słownikowe

6.1 Statyczne kodowanie słownikowe

Zawczasu określamy jakiś słownik słów. Następnie przypisujemy każdemu słowu kod binarny. W ten sposób kodujemy cały tekst. Takie kodowanie ma sporo wad, głównie związanych z koniecznością przesyłania słownika oraz z słabą odpornością na błędy i zmienność danych wejściowych.

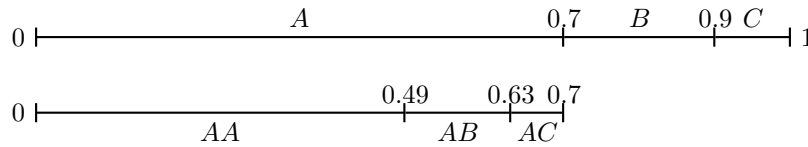


Diagram 3: Wizualizacja kodowania arytmetycznego

6.2 LZ77

Słownikiem jest zakodowana/odkodowana część tekstu. W ten sposób jesteśmy bardzo elastyczni w zakresie zmiany danych wejściowych, oraz nie musimy przesyłać słownika. Kodem jest trójka (o, l, k) gdzie o to przesunięcie, l to długość, a k to kolejny znak. W ten sposób $(0, 0, n), (1, 1, k)$ dekoduje się jako "nnk". Proces kodowania jest parametryzowany n i m , gdzie $o < n$ i $l < m$.

6.3 LZ78

Istnieje osobny słownik, do którego trafiają kolejne słowa. Podczas kodowania kolejno szukamy w słowniku najdłuższego słowa, które jest prefiksem ciągu wejściowego. Jeśli nic nie znajdziemy to dodajemy pierwszą literę do słownika, lecz jeśli znajdziemy taki prefiks, to kodujemy go jako indeks w słowniku, wraz z kodem następnej litery. Zatem kod $(0, k), (1, a)(2, b)$ oznacza "kkakab", a słownik s zawiera $s(1) = k, s(2) = ka, s(3) = kab$.

6.4 LZW

Ta wersja algorytmu pozbywa się drugiego elementu pary z kodowania LZ78. Z kolei potrzebny jest słownik początkowy zawierający wszystkie możliwe symbole. Poza tą małą różnicą, algorytm jest identyczny z LZ78. Zatem ze słownikiem s gdzie $s(1) = a, s(2) = b, s(3) = c$, kod: 34 znaczy "cbcb", ponieważ $s(4) = cb$ po pierwszym kroku.

7 bzip2

Mając blok danych o długości n , tworzymy wszystkie n rotacji tego bloku. Następnie sortujemy je leksykograficznie. W ten sposób otrzymujemy blok transformowany.

0	e	l	l	o	h
1	h	e	l	l	o
2	l	l	o	h	e
3	l	o	h	e	l
4	o	h	e	l	l

Tabela 1: Przykład bloku transformowanego dla słowa "hello"

Na podstawie tej tabeli zapisujemy numer wiersza, w którym znajduje się oryginalne słowo, oraz ostatnią kolumnę. W ten sposób uzyskujemy kod 1, "hoell". Mając tylko te dane, jesteśmy bardzo łatwo w stanie odtworzyć oryginalne słowo. Najpierw sortujemy nasz kod leksykograficznie, zapamiętując indeksy.

0	1	2	3	4
e	h	l	l	o
2	0	3	4	1

Tabela 2: Tabela dekodowania dla kodu 1, "hoell"

Mając taką tabelę, następnie konstruujemy ciąg, traktując tabelę jak permutację, zaczynając od indeksu zawartego w kodzie. W naszym przypadku powstaje permutacja cykliczna $(1, 0, 2, 3, 4)$. Wykorzystując tę permutację, odtwarzamy oryginalne słowo.