STUDIA INFORMATICA 2003
Volume 24 Number 1 (52)

Roman STAROSOLSKI
Politechnika Śląska, Instytut Informatyki

ALGORYTMY BEZSTRATNEJ KOMPRESJI DANYCH

Streszczenie. W artykule, po omówieniu podstawowych pojęć i metod dotyczących bezstratnej kompresji danych, przedstawiono podstawowe uniwersalne algorytmy kompresji, które, chociaż uznawane już za klasyczne, są nadal powszechnie stosowane. Omówiono algorytmy: Huffmana, kodowania arytmetycznego, kodowania długości sekwencji oraz wybrane algorytmy słownikowe. Omówiono również istotne warianty ww. algorytmów oraz przedstawiono przykłady ilustrujące ich działanie.

Słowa kluczowe: uniwersalne algorytmy kompresji, bezstratna kompresja danych, kodowanie, teoria informacji.

LOSSLESS DATA COMPRESSION ALGORITHMS

Summary. In this paper, after presenting basic definitions and methods concerning lossless data compression, we describe basic universal lossless data compression algorithms that although regarded as classical, are still commonly used. We describe: Huffman Coding, Arithmetic Coding, Run Length Encoding and selected dictionary compression algorithms. We also describe important variants and present examples of the above-mentioned algorithms.

Keywords: universal compression algorithms, lossless data compression, coding, information theory.

1. Wprowadzenie

Niniejszy artykuł zawiera przegląd podstawowych pojęć, metod i algorytmów z zakresu bezstratnej kompresji danych. Problematyka omawiana w artykule jest bardzo obszerna. Z tego powodu omawiając algorytmy skoncentrowano się na przedstawieniu ich istoty, nie dążąc do uzyskania takiego poziomu szczegółowości, jak w niektórych pozycjach z cytowanej literatury. W artykule przedstawiono algorytmy obecnie stosowane, które

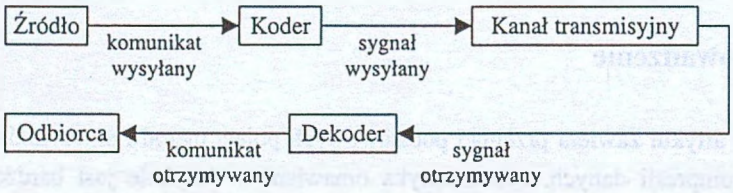
uznawane są już za klasyczne. Nie omawiano algorytmów, które wyszły z użycia, np. kodowania Shannona-Fano, które zostało zastąpione przez algorytm Huffmana. Przedstawiono algorytmy uniwersalne, tj. przeznaczone do kompresji dowolnych klas danych. Przedstawione algorytmy są nadal powszechnie stosowane, a w algorytmach nowoczesnych oraz algorytmach specjalizowanych dla konkretnych klas danych (np. obrazów) występują jako jeden z elementów większego i bardziej złożonego algorytmu nadrzędnego. Algorytmy nowoczesne są tematem oddzielnego opracowania, do którego artykuł niniejszy stanowi wprowadzenie [22].

Problematyka dotycząca algorytmów kompresji stratnej oraz specjalizowanych algorytmów kompresji konkretnych klas danych, takich jak np. obrazy, wykracza poza ramy niniejszego artykułu. Zagadnienia te omówione są obszernie w książkach [1, 15, 17, 18, 26]. Przegląd algorytmów i standardów bezstratnej kompresji obrazów jest tematem oddzielnego artykułu [21].

Układ dalszej części niniejszego artykułu jest następujący: w punkcie 2 znajdują się pojęcia podstawowe dla bezstratnej kompresji danych, zarówno pojęcia z zakresu teorii informacji znajdujące zastosowanie w kompresji danych, jak i te dotyczące wyłącznie kompresji. W kolejnych punktach omówiono: algorytm Huffmana oraz jego udoskonalenia (punkt 3); algorytm kodowania arytmetycznego oraz problemy występujące w praktycznych realizacjach tego algorytmu (4); algorytm kodowania długości sekwencji (5); algorytmy słownikowe LZ77, LZ78 oraz ich udoskonalenia (6). Artykuł kończy krótkie podsumowanie.

2. Pojęcia podstawowe

Kompresja jest szczególnym rodzajem kodowania. W ogólnym przypadku (rys. 1) komunikat, czyli dane generowane przez źródło informacji, przekształcany jest przez koder do postaci sygnału umożliwiającego jego transmisję przez kanał transmisyjny. Po opuszczeniu kanału następuje dekodowanie sygnału do postaci komunikatu, w jakiej otrzymuje ją odbiorca.



Rys. 1. Transmisja danych
Fig. 1. The data transmission

Naturalne źródła informacji to zazwyczaj źródła ciągłe (analogowe), na przykład człowiek przekazujący innemu człowiekowi jakieś informacje za pośrednictwem dźwięków mowy, lub układ chmur na niebie pozwalający meteorologowi przewidywać pogodę. W odróżnieniu od powyższego, źródło dyskretne generuje ciąg symboli z pewnego alfabetu, który będzie określany mianem alfabetu źródła. Jeżeli sygnał z kodera nie zmienia się po opuszczeniu kanału, to taki kanał nazywamy kanałem bezszumowym, w przeciwnym razie mamy do czynienia z kanałem z szumem. W przypadku kanału bezszumowego kodowanie nazywamy bezstratnym, gdy komunikat ze źródła po zakodowaniu, transmisji przez kanał bezszumowy i dekodowaniu nie ulega modyfikacji. Jeżeli natomiast komunikat otrzymywany z dekodera nie jest identyczny z komunikatem wysyłanym przez źródło, to mówimy o kodowaniu ze stratą, bądź kodowaniu stratnym. W niniejszym artykule przyjmuje się, że źródło jest dyskretne, kodowanie bezstratne, a kanał bezszumowy.

Kompresję bezstratną można zdefiniować jako kodowanie bezstratne w taki sposób, aby minimalizować długość komunikatu po zakodowaniu.

Kod to zbiór słów kodowych przyporządkowanych ciągom symboli alfabetu źródła bądź poszczególnym symbolom tego alfabetu. Słowa kodowe to ciągi symboli pewnego alfabetu, który będzie określany mianem alfabetu kodu. Najczęściej stosowane są kody binarne. Kod nazywamy binarnym, jeżeli alfabet kodu to $\{0, 1\}$.

Przyjmijmy następujące oznaczenia:

- $S = \{s_0, s_1, \ldots, s_{n-1}\}$ jest alfabetem źródła, s_i jest symbolem alfabetu źródła, $\|S\| = n$ jest rozmiarem alfabetu źródła. Jeżeli istotna jest wartość liczbowa przypisana danemu symbolowi s_i, to przyjmuje się, że jest to nieujemna liczba całkowita i, $i < \|S\|$.
- $P = \{p_0, p_1, \ldots, p_{n-1}\}$ jest rozkładem prawdopodobieństwa symboli alfabetu źródła. Wygenerowanie przez źródło symbolu s_i to zdarzenie zachodzące z prawdopodobieństwem p_i. W zależności od przyjętego modelu źródła, $p_i \in P$ może być stałe dla całego komunikatu, bądź zmieniać się w zależności od kontekstu, w jakim występuje symbol s_i, i położenia symbolu w komunikacie.
- $K = \{\kappa_0, \kappa_1, \ldots, \kappa_{n-1}\}$ jest kodem przyporządkowującym poszczególnym symbolom alfabetu źródła słowa kodowe tak, że symbolowi s_i odpowiada słowo kodowe κ_i.

Przyjmijmy następujące definicje:

- konkatenacja $A \mid D$ ciągów $A = a_1 a_2 \ldots a_l$ i $D = d_1 d_2 \ldots d_h$, gdzie $l \geq 0$, $h \geq 0$, to ciąg $a_1 a_2 \ldots a_l d_1 d_2 \ldots d_h$ o długości równej $l + h$;
- ciąg D_1 jest przedrostkiem ciągu D, jeżeli istnieje taki ciąg D_2, że $D = D_1 \mid D_2$;

- ciąg D_1 jest przedrostkiem symbolu s w ciągu D, jeżeli istnieją takie ciągi D_0 i D_2, że $D = D_0 \mid D_1 \mid s \mid D_2$;
- ciąg D_2 nazywamy przyrostkiem ciągu D, jeżeli istnieje taki ciąg D_1, że $D = D_1 \mid D_2$.

Jeżeli wszystkie słowa kodowe danego kodu składają się z takiej samej liczby symboli alfabetu kodu, to jest to tzw. kod stałej długości. Kody stałej długości przyporządkowują słowa kodowe symbolom alfabetów o ograniczonej (skończonej) liczbie symboli. Na przykład, kod binarny stałej długości K_n^b służy do kodowania nieujemnych liczb całkowitych mniejszych od n. Kod ten przyporządkowuje liczbom $i \in \{0, 1, \ldots, n-1\}$ słowa kodowe będące ciągami bitów $\kappa_i = b_{N-1} \ldots b_1 b_0$, gdzie $N = \lceil \log_2 n \rceil$, $b_j \in \{0, 1\}$ takie, że $2^0 b_0 + 2^1 b_1 + \ldots + 2^{N-1} b_{N-1} = i$.

W przypadku gdy długości słów kodowych różnią się, to jest to kod zmiennej długości. Kody zmiennej długości przyporządkowują słowa kodowe symbolom alfabetów o ograniczonej, bądź nieskończonej liczbie symboli. Na przykład, kod unarny określony jest dla zbioru nieujemnych liczb całkowitych. Kod unarny przyporządkowuje liczbie i słowo kodowe składające się z i jedynek zakończonych zerem (spotyka się również definicje i-tego słowa kodu jako ciągu i zer zakończonych jedynką).

Kod jest jednoznacznie dekodowalny, gdy każdy ciąg słów kodowych kodu można zdekodować jednoznacznie, tzn. istnieje dokładnie jeden sposób podziału ciągu słów kodowych na oddzielne słowa kodowe. O kodzie mówimy, że jest przedrostkowy, gdy żadne ze słów kodowych zawartych w danym kodzie nie jest przedrostkiem innego słowa kodowego w tym kodzie. Każdy kod przedrostkowy jest jednoznacznie dekodowalny. Kod K jest kompletny, gdy fakt, że dany ciąg D jest przedrostkiem słowa kodowego w K, implikuje to, że ciąg $D' = D \mid \alpha$, gdzie α jest symbolem z alfabetu kodu K, jest przedrostkiem słowa kodowego w kodzie K albo innym słowem kodowym w kodzie K.

Oczekiwana średnia długość L_K kodu K dla rozkładu prawdopodobieństwa symboli P wynosi:

$$L_K = \sum_{i=0}^{n-1} p_i l_i^K ,$$

gdzie l_i^K to długość słowa kodowego przyporządkowanego przez kod K symbolowi s_i. Dany kod jest dla danego rozkładu prawdopodobieństwa P optymalny, gdy średnia długość tego kodu jest najmniejsza spośród wszystkich kodów dla rozkładu P. Dla danego P może istnieć więcej niż jeden kod optymalny. Optymalny kod przedrostkowy jest kompletny.

Liczba bitów, jakiej należy użyć do zakodowania symbolu s_i, tak aby zminimalizować długość komunikatu po zakodowaniu, to ilość autoinformacji, $I(s_i)$, stowarzyszonej ze zdarzeniem polegającym na wygenerowaniu przez źródło symbolu s_i. Ilość autoinformacji $I(s_i)$ zależy wyłącznie od prawdopodobieństwa p_i wygenerowania symbolu s_i:

$$I(s_i) = -\log_2(p_i).$$

Ilość autoinformacji przypadającą na pojedynczy symbol generowany przez źródło z rozkładem prawdopodobieństwa P nazywamy entropią H rozkładu prawdopodobieństwa. Wartość entropii zależy wyłącznie od rozkładu prawdopodobieństwa symboli alfabetu:

$$H = \sum_{i=0}^{n-1} p_i I(s_i) = -\sum_{i=0}^{n-1} p_i \log_2(p_i).$$

W teorii informacji — dziedzinie wiedzy rozwiniętej w latach 40. przez Claude E. Shannona [16] — entropia to pojęcie fundamentalne. Entropia jest miarą niepewności, bądź ilości informacji, jaką niesie ze sobą wygenerowanie symbolu przez źródło. W przypadku gdy źródło jest bezpamięciowe, czyli gdy kolejne symbole generowane są przez źródło niezależnie, H jest minimalną średnią długością kodu, jaką może uzyskać koder.

Na podstawie entropii określa się jakość kodów i procesu kodowania. Najczęściej spotykane miary jakości to nieefektywność F_K kodu K, której jednostką jest bit na symbol oraz efektywność E_K kodu K, która zazwyczaj wyrażana jest w procentach:

$$F_K = L_K - H, \qquad E_K = \frac{H}{L_K} \cdot 100\%.$$

W praktyce często znamy komunikat wygenerowany przez źródło (na przykład plik z tekstem), ale nie dysponujemy modelem źródła. W takiej sytuacji, po arbitralnym założeniu modelu źródła, empirycznie wyznacza się entropię ciągu na podstawie dostępnego ciągu. Entropia H_D danego ciągu D określa ilość autoinformacji przypadającą na pojedynczy symbol ciągu. Niech: $D = d_1 d_2 \ldots d_u$, $u > 0$, $d_i \in S$, wtedy:

$$H_D = -\frac{1}{u}\sum_{i=1}^{u} \log_2(p_{d_i}^i),$$

gdzie u to długość ciągu, a $p_{d_i}^i$ to prawdopodobieństwo wystąpienia symbolu d_i na i-tej pozycji w ciągu D. Sposób wyznaczania prawdopodobieństwa $p_{d_i}^i$ zależy od przyjętego modelu źródła. Zazwyczaj jest to prawdopodobieństwo warunkowe wyznaczone na podstawie znajomości skończonej i niewielkiej liczby symboli poprzedzających dany symbol — liczba symboli determinuje rząd entropii. Jeżeli przyjmiemy, że źródło jest bezpamięciowe, to prawdopodobieństwo wystąpienia danego symbolu wyznaczane jest niezależnie od pozostałych symboli i mówimy wtedy o entropii rzędu zerowego.

Współczynnik kompresji jest miarą pozwalającą oceniać efekty wykonywania algorytmów kompresji. W literaturze spotyka się wiele różnych definicji tego współczynnika. Współczynnik kompresji wyrażany jest w bitach na symbol: $\frac{z}{u}$, gdzie u oznacza liczbę symboli kompresowanego komunikatu, $u > 0$, a z liczbę bitów komunikatu po zakodowaniu. Tak zdefiniowany współczynnik nazywany jest również średnią bitową. Dla tak zdefiniowanego współczynnika kompresja jest tym lepsza, im wartość współczynnika mniejsza. W przypadku kompresji obrazów kodowany komunikat składa się z pikseli. Współczynnik kompresji obrazu wyrażany jest w bitach na piksel [bpp] (ang. bits per pixel). Inne miary efektów wykonywania algorytmów kompresji to stopień kompresji: $\frac{uN}{z}$, gdzie $N = \lceil \log_2 |S| \rceil$ oznacza liczbę bitów potrzebną do zakodowania symboli alfabetu źródła kodzie binarnym stałej długości, oraz procent kompresji: $\left(1 - \frac{z}{uN}\right) \cdot 100\%$. Jeżeli w wyniku kompresji zachodzi $uN < z$, to mówimy, że nastąpiła ekspansja danych. Ekspansja wyrażana jest w bitach na symbol: $\frac{z}{u} - N$, bądź procentowo: $\left(\frac{z}{uN} - 1\right) \cdot 100\%$.

W ogólnym przypadku do przeprowadzenia kompresji konieczne jest określenie kodu dla komunikatu generowanego przez źródło. Kod jest konstruowany na podstawie charakterystyki kompresowanego komunikatu tak, aby zminimalizować długość komunikatu po zakodowaniu. Informacje o charakterystyce kompresowanego komunikatu pochodzą z tzw. modelu danych. Ze względu na sposób budowania modelu danych, algorytmy kompresji, określane również mianem kompresorów, możemy podzielić na:

- stałe — w przypadku stałych algorytmów kompresji zakłada się, że komunikat ma pewną z góry określoną charakterystykę i stosuje się założony a priori model danych, bez względu na rzeczywistą treść komunikatu. Zastosowany model danych zbudowany jest, jeszcze przed rozpoczęciem przetwarzania komunikatu, dla całej klasy komunikatów. Jest to więc model pewnej klasy komunikatów, a nie konkretnego komunikatu. To, jak dobrze model nadawać się będzie dla konkretnego komunikatu, zależy od tego, w jakim stopniu charakterystyka tego komunikatu zbliżona będzie do charakterystyki klasy komunikatów, dla której zbudowano model danych;
- statyczne — model danych buduje się na podstawie znajomości całego komunikatu, który będzie podlegał kompresji. Jest to więc model konkretnego komunikatu, stały dla tego komunikatu. Dopiero po wygenerowaniu całego komunikatu przez źródło i przeanalizowaniu całości komunikatu można rozpocząć

kodowanie, zazwyczaj czytając po raz drugi komunikat i kodując jego kolejne symbole z użyciem modelu danych. Aby umożliwić poprawne zdekodowanie komunikatu, do dekodera oprócz zakodowanego komunikatu należy przekazać model danych lub informacje wystarczające do zrekonstruowania modelu;

- adaptacyjne — tutaj model danych wyznaczany jest na podstawie informacji uzyskanych na podstawie analizy wcześniej zakodowanej części komunikatu. Każdorazowo po zakodowaniu symbolu źródła (bądź obiektu, czyli ciągu symboli źródła) następuje uaktualnienie informacji o charakterystyce kompresowanego komunikatu przechowywanych w modelu danych. Jest to więc budowany dynamicznie, przyrostowy model konkretnego komunikatu. W przeciwieństwie do algorytmów statycznych, kodowanie przeprowadzane jest w miarę generowania symboli przez źródło, a do dekodera wystarczy przekazać sam zakodowany komunikat. Dekoder gromadzi informacje o charakterystyce komunikatu analogicznie do kodera na podstawie już dostępnej części komunikatu.

Jeżeli złożoność kompresji i dekompresji dla danego algorytmu są zbliżone, to taki algorytm nazywamy symetrycznym, w przeciwnym razie algorytm jest asymetryczny, przy czym z reguły mniejsze są złożoności dekompresji. Asymetryczność algorytmu jest cechą pożądaną w niektórych zastosowaniach, na przykład, gdy komunikat generowany przez źródło jest na bieżąco kompresowany, transmitowany do odbiorcy, a następnie dekompresowany za pomocą dekodera dysponującego zasobami istotnie mniejszymi od zasobów kodera.

W algorytmach kompresji do efektywnego zakodowania komunikatu wykorzystuje się pewne cechy charakterystyczne tego komunikatu. W przypadku najprostszych algorytmów są to proste do zidentyfikowania cechy, takie jak występowanie ciągów powtarzających się symboli (punkt 5 niniejszego artykułu). Spośród wielu bezstratnych algorytmów kompresji danych najlepiej poznane są dwie grupy algorytmów: algorytmy statystyczne i słownikowe.

W algorytmach statystycznych, na podstawie znajomości rozkładu prawdopodobieństwa symboli alfabetu źródła koduje się poszczególne symbole komunikatu tak, aby liczba bitów odpowiadająca danemu symbolowi w zakodowanym komunikacie zbliżona była do autoinformacji przyporządkowanej temu symbolowi. Informacje pozwalające na określenie rozkładu prawdopodobieństwa symboli alfabetu źródła gromadzone są w modelu danych. Jeżeli prawdopodobieństwo wystąpienia dla danego symbolu komunikatu wyznacza się niezależnie od pozostałych symboli komunikatu, to jest to model danych rzędu zerowego. W modelach rzędów niezerowych przechowuje się informacje pozwalające określić prawdopodobieństwo warunkowe. W tym przypadku rząd modelu danych określa, ile symboli (określanych mianem kontekstu) bierze udział w wyznaczeniu prawdopodobieństwa

wystąpienia danego symbolu w komunikacie. Zazwyczaj kontekstem dla danego symbolu komunikatu jest pewna liczba symboli bezpośrednio go poprzedzających. W najprostszych modelach danych pamięta się rozkład prawdopodobieństwa symboli alfabetu źródła, bardziej złożone modele danych omówiono w [22]. Algorytm służący do zakodowania danego symbolu komunikatu dla danego rozkładu prawdopodobieństwa symboli alfabetu źródła określany jest mianem kodera entropijnego. Najczęściej stosowane kodery entropijne to algorytm kodowania arytmetycznego (punkt 4) i kodowanie z zastosowaniem kodów Huffmana (punkt 3). Dla niektórych klas danych, np. obrazów, stosowane są z powodzeniem również proste kodery entropijne, w których kodowanie polega na użyciu kodu, wybranego z pewnej rodziny kodów (np. z rodziny Golomba). Dla danego symbolu komunikatu, na podstawie informacji zgromadzonych w modelu danych, dokonuje się wyboru kodu z rodziny. Koder entropijny wyprowadza jedynie słowo kodowe przyporządkowane danemu symbolowi przez wybrany kod. W takim przypadku, mimo że rodzina kodów została wyznaczona niezależnie od postaci konkretnego komunikatu, algorytm jest adaptacyjny, gdy użyty model danych jest adaptacyjny. Przykłady takich algorytmów omówiono w [21].

W algorytmach słownikowych rolę modelu danych pełni tzw. słownik. Słownik pozwala na wyróżnianie fraz, czyli ciągów symboli alfabetu źródła, których wystąpienia w komunikacie się spodziewamy. Kompresor po napotkaniu w ciągu wejściowym frazy, która już się znajduje w słowniku, wyprowadza na wyjście zamiast frazy jej indeks. Większość algorytmów słownikowych jest asymetrycznych. Najczęściej stosowane są następujące algorytmy słownikowe:

- algorytm LZ77 [27] i pochodne (np. LZSS), gdzie rolę słownika pełni bufor z określoną liczbą ostatnio przetworzonych symboli. Aby jednoznacznie zidentyfikować frazę, trzeba podać zarówno jej pozycję w buforze, jak i jej długość;
- algorytm LZ78 [28] i pochodne (np. LZW), gdzie korzysta się ze słownika bardziej odpowiadającego intuicyjnemu rozumieniu słowa „słownik" — do jednoznacznego zidentyfikowania frazy wystarczy znajomość jej indeksu.

3. Algorytm Huffmana

Opublikowany w roku 1952 przez Davida Huffmana [4] algorytm służy do wyznaczania kodu dla danego rozkładu prawdopodobieństwa symboli. Kod wyznaczony z użyciem tego algorytmu zwany jest kodem Huffmana.

Kod Huffmana jest optymalny w klasie kodów przedrostkowych. Nieefektywność F_{Huff} kodu Huffmana jest z góry ograniczona i wynosi:

$$F_{Huff} \leq p_{max} + \log_2\left(\frac{2 \cdot \log_2 e}{e}\right) \approx p_{max} + 0.086,$$

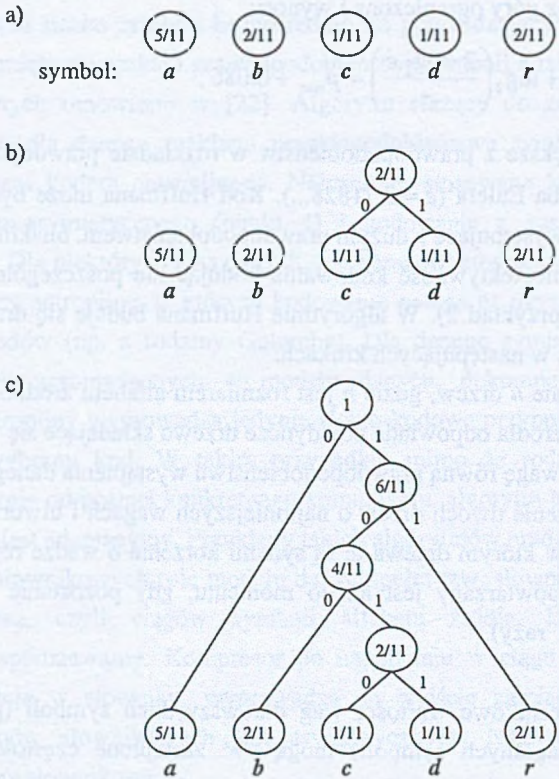
gdzie p_{max} to największe z prawdopodobieństw w rozkładzie prawdopodobieństwa symboli alfabetu, a e to liczba Eulera ($e \approx 2.71828...$). Kod Huffmana może być nieefektywny, gdy kodujemy symbole występujące z dużym prawdopodobieństwem, bliskim 1. W takiej sytuacji można ograniczyć nieefektywność kodowania kodując nie poszczególne symbole alfabetu, lecz ciągi symboli (przykład 2). W algorytmie Huffmana buduje się drzewo binarne, zwane drzewem Huffmana, w następujących krokach:

1. Utworzenie n drzew, gdzie n jest rozmiarem alfabetu źródła. Każdemu z symboli alfabetu źródła odpowiada pojedyncze drzewo składające się wyłącznie z korzenia i mające wagę równą prawdopodobieństwu wystąpienia danego symbolu.
2. Wyznaczenie dwóch drzew o najmniejszych wagach i utworzenie z nich nowego drzewa, w którym drzewa te są synami korzenia o wadze równej sumie ich wag. Krok 2 powtarzany jest aż do momentu, gdy pozostanie tylko jedno drzewo (tj. $n-1$ razy).

W kroku 1. początkowe wartości wag dla wszystkich symboli (prawdopodobieństwa wystąpienia poszczególnych symboli) mogą być zastąpione częstościami występowania poszczególnych symboli w komunikacie, dla którego budowane jest drzewo Huffmana, lub liczbami wystąpień poszczególnych symboli w komunikacie.

W wyniku wykonania algorytmu otrzymuje się drzewo binarne, w którym każdemu z liści odpowiada pojedynczy symbol alfabetu źródła. Słowo kodowe kodu Huffmana dla danego symbolu znajduje się przechodząc ścieżką od korzenia drzewa Huffmana do liścia odpowiadającego temu symbolowi (i-ty symbol słowa kodowego ma wartość 0, jeżeli i-ta krawędź ścieżki prowadzi do lewego syna i-tego węzła, a 1 — jeżeli do prawego).

Przykład 1. Budujemy drzewo Huffmana dla komunikatu ***abracadabra***. Alfabet źródła to $\{a, b, c, d, r\}$. W pierwszym kroku tworzonych jest 5 drzew (rys. 2.a). Wagi poszczególnych drzew to prawdopodobieństwa występowania odpowiadających im symboli wyznaczone na podstawie liczby wystąpień tych symboli w kodowanym komunikacie. Na rys. 2.b widnieje efekt pierwszego przeprowadzenia kroku 2. Rys. 2.c przedstawia gotowe drzewo Huffmana. Kod Huffmana dla tego drzewa umieszczono w tabeli 1.



Rys. 2. Budowanie drzewa Huffmana komunikatu *abracadabra*
Fig. 2. Building the Huffmann tree for the message *abracadabra*

Tabela 1
Kod Huffmana dla drzewa z rys. 2.c

Symbol	Słowo kodowe
a	0
b	100
c	1010
d	1011
r	11

Komunikat *abracadabra*, który do zakodowania w kodzie binarnym stałej długości wymaga 33 bitów, po zakodowaniu kodem Huffmana ma długość 23 bitów (01001101010010110100110).

Słowo kodowe przyporządkowane symbolowi przez binarny kod przedrostkowy ma długość przynajmniej jednego bitu. Kod taki jest nieefektywny wtedy, gdy prawdopodobieństwo wystąpienia jednego z symboli jest bliskie 1. W takiej sytuacji długość słowa kodowego tego symbolu jest wielokrotnie większa od przyporządkowanej mu autoinformacji, której wartość jest bliska 0, a co za tym idzie — średnia długość kodu jest wielokrotnie większa od entropii rozkładu prawdopodobieństwa symboli. Kody przedrostkowe nie nadają się również do kompresji komunikatów ze źródeł binarnych. Dla źródeł binarnych średnia długość optymalnego kodu przedrostkowego będzie wynosiła 1, czyli tyle, ile długość kodu binarnego stałej długości. Efektywne zastosowanie kodów Huffmana dla komunikatów takich, jak wyżej wspomniane, jest możliwe, jeżeli kodować będziemy nie poszczególne symbole, lecz ciągi symboli (na przykład pary symboli). Metodę tę ilustruje poniższy przykład.

Przykład 2. Kodujemy komunikat generowany przez źródło binarne o alfabecie źródła $S = \{0, 1\}$. $P = \{0.9, 0.1\}$ jest rozkładem prawdopodobieństwa symboli alfabetu źródła. Entropia rozkładu prawdopodobieństwa wynosi 0.467. Średnia długość kodu Huffmana dla alfabetu binarnego wynosi 1, gdyż niezależnie od rozkładu prawdopodobieństwa symboli kod ten przyporządkowuje obu symbolom alfabetu binarnego słowa kodowe o długości 1. Efektywność kodu Huffmana dla tego źródła jest niewielka, ponieważ wynosi niespełna 47 %. Jeżeli kodować będziemy pary symboli (tabela 2), to efektywność kodu wzrośnie do około 72 %.

Tabela 2

Kod Huffmana dla par symboli

Para	Prawdopodobieństwo	Słowo kodowe
00	0.81	0
01	0.09	10
10	0.09	110
11	0.01	111

Modelem danych w algorytmie stosującym kody Huffmana najczęściej jest tablica liczb wystąpień poszczególnych symboli alfabetu. Algorytm Huffmana znajduje zastosowanie przede wszystkim w uniwersalnych kompresorach statycznych. W kompresorze statycznym skompresowany komunikat musi, oprócz zakodowanego komunikatu, zawierać opis drzewa Huffmana lub, co w praktyce zajmuje mniej miejsca, informacje pozwalające na zrekonstruowanie tego drzewa przez dekoder.

Kody Huffmana stosuje się również w stałych algorytmach kompresji. Przykładem może być kompresor przeznaczony do kodowania tekstów jakiegoś języka naturalnego bądź języka programowania. Na przykład, w kompresorze programów napisanych w języku „C" powinno się użyć kodu Huffmana wyznaczonego na podstawie analizy reprezentatywnego zbioru komunikatów, w tym przypadku programów w języku „C". Analizując odpowiednio obszerny zbiór komunikatów, można zaobserwować pewne cechy charakterystyczne, takie jak na przykład występowanie ciągów symboli tworzących słowa kluczowe języka „C", wraz z prawdopodobieństwem ich występowania. Cechy te byłyby trudne do zidentyfikowania w przypadku pojedynczego, zwłaszcza krótkiego, komunikatu i trudno by było efektywnie przekazać je do dekodera w przypadku algorytmu statycznego.

Dla kompresora adaptacyjnego budowanie drzewa Huffmana każdorazowo po wyprowadzeniu słowa kodowego i aktualizacji modelu danych jest możliwe, lecz zbyt czasochłonne, by nadawało się do zastosowania w praktyce. Tak zwane algorytmy dynamicznego kodowania Huffmana [3, 8, 24, 25] opisują sposób modyfikacji już istniejącego drzewa Huffmana, po zakodowaniu danego symbolu i zmianie wag przypisanych poszczególnym symbolom. Modyfikacja przeprowadzana jest w taki sposób, by po jej wykonaniu uzyskać drzewo Huffmana dla nowego rozkładu prawdopodobieństwa symboli alfabetu.

4. Kodowanie arytmetyczne

Algorytm kodowania arytmetycznego to koder entropijny generujący kod optymalny w klasie kodów jednoznacznie dekodowalnych. Za twórców idei kodowania arytmetycznego uważa się C.E. Shannona i P. Eliasa. Implementowalny algorytm kompresji oparty na tej idei został zaprezentowany po raz pierwszy w roku 1975 przez J. Rissanena. Od tego czasu algorytm kodowania arytmetycznego jest przedmiotem wielu badań i udoskonaleń [5, 6, 7, 9, 11, 12, 13, 14, 19].

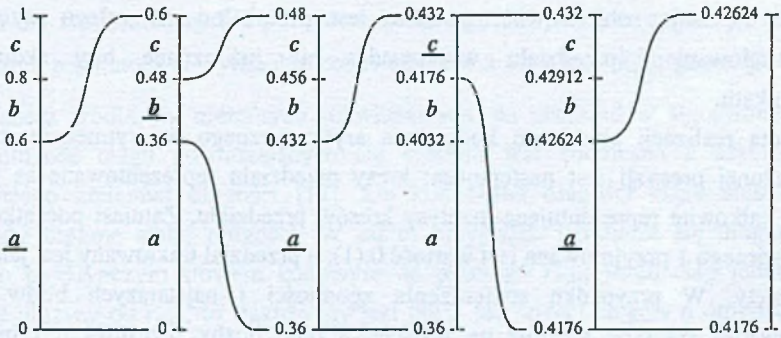
W algorytmie kompresji arytmetycznej koduje się cały komunikat za pomocą pojedynczej liczby rzeczywistej z pewnego lewostronnie domkniętego przedziału zawartego w przedziale [0, 1). Kodując kolejne symbole komunikatu zawęża się stopniowo początkowy przedział [0, 1). Dla danego symbolu z komunikatu przedział dzieli się na podprzedziały o długościach wprost proporcjonalnych do prawdopodobieństw przypisanych poszczególnym symbolom alfabetu źródła przez model danych, po czym wybiera się przedział odpowiadający temu symbolowi. Po przetworzeniu wszystkich symboli komunikatu wyprowadza się dowolną liczbę z wyznaczonego przedziału. Jeżeli długość komunikatu nie jest znana dekoderowi,

to należy ją uprzednio oddzielnie przetransmitować, bądź uzupełnić alfabet źródła o symbol oznaczający koniec komunikatu.

Kompresor, w którym korzysta się z kodowania arytmetycznego, pozbawiony jest wad kompresora stosującego kod przedrostkowy do kodowania symboli alfabetu binarnego lub alfabetu źródła, w którym jeden z symboli ma duże prawdopodobieństwo wystąpienia. Wielką zaletą statystycznego algorytmu kompresji, w którym koderem entropijnym jest algorytm kodowania arytmetycznego, jest oddzielenie modelu danych od kodowania, dzięki czemu mamy łatwość stosowania różnych modeli danych. W zależności od przyjętego modelu danych kompresor może być adaptacyjny, statyczny bądź stały.

Przykład 3. Kompresujemy komunikat *abaca*, alfabet źródła to $\{a, b, c\}$. Używamy stałego modelu danych przypisującego symbolom alfabetu, niezależnie od położenia symbolu w komunikacie prawdopodobieństwa $P=\{0.6, 0.2, 0.2\}$. Przyjmujemy, iż długość komunikatu jest znana dekoderowi. Działanie algorytmu ilustrują rys. 3 i tabela 3.

Rozpoczynamy kompresję od podziału przedziału [0, 1) na podprzedziały odpowiadające symbolom alfabetu źródła: [0, 0.6) dla *a*, [0.6, 0.8) dla *b* i [0.8, 1) dla *c*. Pobieramy pierwszy symbol komunikatu, *a*, i zawężamy przedział do podprzedziału odpowiadającego temu symbolowi, tj. [0, 0.6). Operacje podziału przedziału na podprzedziały, pobrania kolejnego symbolu komunikatu i zawężenia przedziału do podprzedziału odpowiadającego pobranemu symbolowi powtarzamy aż do wyczerpania symboli komunikatu. Kolejnym symbolem komunikatu jest *b*. Ponieważ nie przetworzyliśmy jeszcze wszystkich symboli komunikatu, to otrzymany w poprzednim kroku przedział ponownie dzielimy na podprzedziały o długościach wprost proporcjonalnych do prawdopodobieństw występowania symboli: [0, 0.36) dla *a*, [0.36, 0.48) dla *b* i [0.48, 0.8) dla *c*. Pobieramy kolejny symbol komunikatu, *b*, i zawężamy przedział do [0.36, 0.48).



Rys. 3. Kodowanie arytmetyczne
Fig. 3. The arithmetic coding

Po przetworzeniu całego komunikatu *abaca* otrzymujemy przedział [0.4176, 0.42624). W systemie dwójkowym kres dolny przedziału to 0.01101010110... , a kres górny to 0.01101101000... . Wybieramy liczbę leżącą wewnątrz przedziału 0.011011 i wyprowadzamy znaczące cyfry mantysy, tj. 011011, jako zakodowany komunikat. Już na przykładzie tak krótkiego komunikatu widać, iż kodowanie arytmetyczne przewyższa kodowanie Huffmana, gdyż ten sam komunikat zakodowany kodami Huffmana ma długość 7 bitów.

Tabela 3

Kodowanie arytmetyczne

Przedział	Symbol	Przedziały odpowiadające symbolom		
		a	*b*	*c*
[0, 1)	*a*	**[0, 0.6)**	[0.6, 0.8)	[0.8, 1)
[0, 0.6)	*b*	[0, 0.36)	**[0.36, 0.48)**	[0.48, 0.6)
[0.36, 0.48)	*a*	**[0.36, 0.432)**	[0.432, 0.456)	[0.456, 0.48)
[0.36, 0.432)	*c*	[0.36, 0.4032)	[0.4032, 0.4176)	**[0.4176, 0.432)**
[0.4176, 0.432)	*a*	**[0.4176, 0.42624)**	[0.42624, 0.42912)	[0.42912, 0.432)
[0.4176, 0.432)				

Wyżej przedstawiony algorytm nie nadaje się do bezpośredniej realizacji praktycznej, gdyż stosuje się w nim arytmetykę zmiennopozycyjną o nieograniczonej precyzji. Wymagana precyzja zależna jest od entropii oraz długości komunikatu, co do których nie czynimy żadnych założeń.

W praktycznych realizacjach algorytmów kompresji arytmetycznej wykonuje się obliczenia z zastosowaniem liczb o ograniczonej precyzji. Metoda taka powoduje nieznaczne pogorszenie współczynnika kompresji, gdyż precyzja wyznaczania prawdopodobieństw na podstawie modelu danych oraz podziału przedziału na podprzedziały jest ograniczona. Ponieważ początkowe cyfry rozwinięcia binarnego dolnego i górnego kresu przedziału szybko się stają sobie równe, możliwe jest przeskalowanie całego przedziału. Wraz z przeskalowaniem przedziału wyprowadza się już znane bity skompresowanego komunikatu.

Istota realizacji algorytmu kodowania arytmetycznego w arytmetyce stałopozycyjnej o określonej precyzji jest następująca: kresy przedziału reprezentowane są przez binarne liczby całkowite reprezentujące mantysy kresów przedziału. Zamiast początkowej wartości kresu górnego 1 przyjmowana jest wartość 0.(1), a przedział traktowany jest jako obustronnie domknięty. W przypadku stwierdzenia zgodności *i* najstarszych bitów obu kresów wyprowadza się tych *i* bitów na wyjście, a obie liczby przesuwa o *i* bitów w lewo. W rezultacie *i* najmłodszych bitów kresu dolnego przyjmuje wartość 0, a górnego 1. Przeprowadzając obliczenia w arytmetyce stałopozycyjnej, należy uwzględnić możliwość

pojawienia się niedomiaru, gdy kresy przedziału różnią się nieznacznie, ale początki ich rozwinięć binarnych nie pokrywają się, np. kresy 0.1000000... i 0.0111111... . Konsekwencją zastosowania skończonej dokładności obliczeń jest również niebezpieczeństwo wyznaczenia podprzedziału o długości, która zostanie zaokrąglona do zera. Dlatego też w zależności od precyzji wykonywanych obliczeń nakłada się dolne ograniczenie na minimalne prawdopodobieństwo przyporządkowywane dowolnemu symbolowi alfabetu źródła przez model danych. Przeprowadza się również odpowiednie korekcje kresów przedziału w przypadku stwierdzenia zaistnienia wspomnianego niebezpieczeństwa.

Szczególnym przypadkiem kompresora arytmetycznego jest binarny kompresor arytmetyczny, tj. kompresor kodujący komunikaty ze źródeł binarnych. W praktycznych realizacjach algorytmu kompresji arytmetycznej przyjęcie alfabetu źródła {0, 1} pozwala na daleko idące uproszczenia zarówno kodera, jak i modelu danych. Pierwsze praktyczne implementacje algorytmu kompresji arytmetycznej były binarnymi kompresorami arytmetycznymi.

5. Kodowanie długości sekwencji

Działanie algorytmu kodowania długości sekwencji (ang. Run Length Encoding, RLE) polega na kodowaniu ciągów powtarzających się symboli. Słowem kodowym przyporządkowanym ciągowi składającemu się z r symboli s jest para $<r, s>$. Jeżeli dany symbol s różni się od symbolu poprzedzającego go i symbolu występującego po nim w kompresowanym komunikacie, to kodowany jest jako $<1, s>$. Na przykład, komunikat *aaaaabbbcaaaaaaaaaaccccccccc* zakodowany zostanie za pomocą następującego ciągu słów kodowych: $<5, a><3, b><1, c><10, a><8, c>$.

W ogólnym przypadku para $<r, s>$ wyprowadzana jest jako ¤ r ¤ s, gdzie ¤ jest separatorem dla słów kodowych i pól słowa kodowego. W praktyce nie ma konieczności wyprowadzania separatora ¤. Symbol s kodowany jest z użyciem $K^b_{|S|}$, gdzie $||S||$ oznacza rozmiar alfabetu źródła. W niektórych rozwiązaniach, na przykład w algorytmie CCITT Group 3, długość ciągu powtarzających się symboli jest kodowana z użyciem kodu przedrostkowego zmiennej długości [17]. Do kodowania długości ciągu stosowane są również kody binarne stałej długości. W takim przypadku ogranicza się długość ciągu kodowanego pojedynczym słowem kodowym do pewnego r_{max}; jeżeli ciąg jednakowych symboli jest dłuższy od r_{max}, to traktowany jest jak 2 lub więcej ciągów o długościach nie przekraczających r_{max}. Pomniejszona o 1 długość ciągu r kodowana jest kodem binarnym stałej długości $K^b_{r_{max}}$ (długości 1 ... r_{max} kodowane są jako liczby odpowiednio 0 ... $r_{max} - 1$). Kod RLE skonstruowany w taki sposób jest kodem przedrostkowym.

Kodowanie RLE nadaje się do kompresji danych zawierających ciągi powtarzających się symboli, przykładem takich danych mogą być ciągi pikseli obrazów binarnych lub utworzonych komputerowo rysunków. Zastosowanie algorytmu RLE prowadzi do ekspansji danych, jeżeli udział ciągów powtarzających się symboli w kompresowanym komunikacie jest niewielki. Ekspansję danych ogranicza się dopuszczając, oprócz kodowania ciągów powtarzających się symboli, również kodowanie ciągów różnych symboli.

6. Algorytmy słownikowe

Algorytm LZ77 zaprezentowany został w roku 1977 przez J. Ziva i A. Lempela [27]. W algorytmie tym przechowuje się bufor (tzw. okno) będący konkatenacją bufora słownikowego zawierającego określoną liczbę ostatnio zakodowanych symboli oraz bufora kodowania z pewną liczbą symboli, które dopiero mają być zakodowane. Inicjalizując proces kodowania wypełniamy bufor słownikowy powtórzeniami pierwszego symbolu komunikatu podlegającego kompresji, a bufor kodowania przedrostkiem kodowanego komunikatu, o długości równej długości bufora kodowania. Następnie cyklicznie wyszukujemy najdłuższy przedrostek f bufora kodowania w oknie (tzw. dopasowanie). Poszukiwania rozpoczynamy od końca bufora słownikowego i przesuwamy się w stronę początku okna. Znalezione w ten sposób dopasowanie f może mieć długość większą od długości bufora słownikowego, nie większą jednak od długości okna. Po znalezieniu f wyprowadzamy trójkę $<g_f, l_f, s>$, gdzie g_f oznacza pozycję f w buforze słownikowym, l_f to długość f, a s to symbol, dla którego f jest przedrostkiem w oknie, czyli pierwszy symbol znajdujący się w buforze kodowania za f. Po wyprowadzeniu trójki $<g_f, l_f, s>$ zawartość okna przesuwamy o l_f pozycji w lewo, wprowadzając jednocześnie do bufora kodowania kolejne, nie pobrane jeszcze symbole komunikatu. Zaprezentowany w roku 1982 przez J. Storera i T. Szymańskiego algorytm LZSS [20] to istotne udoskonalenie algorytmu LZ77, polegające na tym, że jeżeli znalezione dopasowanie f jest krótsze od określonego minimum l_{min}, to wyprowadza się parę $<1, s>$, a w przeciwnym przypadku trójkę $<0, g_f, l_f>$. Pseudokod algorytmu LZSS zamieszczono na rys. 4. Pierwsze elementy par i trójek wyprowadzanych w algorytmie LZSS to bity. Dlatego nawet w przypadku, gdy dopasowanie f jest ciągiem pustym, ekspansja przy kodowaniu s wyniesie 1 bit, tj. kilkakrotnie mniej niż dla algorytmu LZ77. Co więcej w przypadku znalezienia odpowiednio długiego przedrostka f nie wyprowadzamy od razu symbolu s następującego po nim, dzięki czemu s może zostać później zakodowany jako pierwszy symbol w dłuższym ciągu.

```
zainicjalizuj okno
while (nie pobrano wszystkich symboli komunikatu) do
    znajdź najdłuższe dopasowanie f bufora kodowania w oknie
    if (l_f < l_min) then
        wyprowadź parę <1, pierwszy symbol z bufora kodowania>
        przesuń zawartość okna o 1 pozycję w lewo
    else
        wyprowadź trójkę <0, g_f, l_f>
        przesuń zawartość okna o l_f pozycji w lewo
    endif
endwhile
```

Rys. 4. Algorytm LZSS
Fig. 4. The LZSS algorithm

Algorytm LZ78 zaprezentowany w roku 1978 przez J. Ziva i A. Lempela [28] można w uproszczeniu przedstawić za pomocą pseudokodu (rys. 5). W algorytmie LZ78 pobiera się kolejne symbole komunikatu budując z nich pewną frazę. Po pobraniu kolejnego symbolu i dodaniu go na końcu już zbudowanej frazy sprawdza się, czy słownik tę frazę zawiera. Jeżeli frazy nie ma w słowniku, to wyprowadzany jest indeks najdłuższej dopasowanej frazy i ostatnio pobrany symbol. Po zakodowaniu frazy i symbolu wstawia się do słownika frazę, dla której operacja szukania zakończyła się niepowodzeniem i rozpoczyna się budowanie nowej frazy od frazy pustej.

```
opróżnij słownik
f := fraza pusta
while (nie pobrano wszystkich symboli komunikatu) do
    pobierz symbol s
    if (f|s znajduje się w słowniku) then
        f := f|s
    else
        wyprowadź parę <indeks frazy f, s>
        wstaw frazę f|s do słownika
        f := fraza pusta
    endif
endwhile
wyprowadź parę <indeks frazy f, s>
```

Rys. 5. Algorytm LZ78
Fig. 5. The LZ78 algorithm

Algorytm LZW (rys. 6) to zaproponowane w roku 1984 przez T.A. Welcha udoskonalenie algorytmu LZ78 polegające na wstępnym wypełnieniu słownika jednoznakowymi frazami z wszystkimi symbolami alfabetu. Dzięki tej modyfikacji wyprowadzane są tylko indeksy fraz, a współczynniki kompresji uzyskiwane przez algorytm LZW są lepsze niż w przypadku algorytmu LZ78.

```
zainicjalizuj słownik
f := fraza pusta
while (nie pobrano wszystkich symboli komunikatu) do
    pobierz symbol s
    if (f|s znajduje się w słowniku) then
        f := f|s
    else
        wyprowadź indeks frazy f
        wstaw frazę f|s do słownika
        f := s
    endif
endwhile
wyprowadź indeks frazy f
```

Rys. 6. Algorytm LZW
Fig. 6. The LZW algorithm

Działanie algorytmu kompresji i dekompresji LZW zilustrowane zostanie na poniższym przykładzie. Przykład ten ilustruje również pewien szczególny przypadek powodujący, iż algorytm dekompresora nie jest trywialnym odtworzeniem operacji wykonywanych przez kompresor. Dekompresor, po otrzymaniu kodu danej frazy, na wyjście wyprowadza odpowiednią frazę ze słownika, ale w stosunku do kompresora nowe frazy do słownika wstawiane są z opóźnieniem. Nowa fraza może być wstawiona do słownika dopiero po otrzymaniu kodu następnej frazy, gdyż składa się ona z frazy aktualnej i pierwszego znaku następnej frazy.

Przykład 4. Algorytmem LZW kodujemy komunikat *barbararabarbarbar*. Alfabet źródła to $\{a, b, r\}$. Wstępnie słownik wypełniamy frazami (w nawiasach podano indeksy fraz): *a* (0), *b* (1) i *r* (2). W pierwszej kolumnie tabeli 4 widnieje pozostała do zakodowania część komunikatu, poszczególne jej fragmenty rozdzielone znakami „|" to odpowiednio: najdłuższy przedrostek znaleziony w słowniku, pierwszy symbol, po pobraniu którego nie powiodła się operacja wyszukania w słowniku, oraz pozostała, jeszcze nie pobierana część komunikatu.

Wspomniany przypadek szczególny występuje tu w momencie (komórki podkreślone w tabeli 4), gdy dekoder otrzymuje indeks frazy 10, której jeszcze nie ma w słowniku. Taka sytuacja ma miejsce jedynie w przypadku, gdy dana fraza jest konkatenacją ostatnio zakodowanej frazy i pierwszego symbolu ostatnio zakodowanej frazy. W przypadku gdy dekoder otrzymuje indeks frazy, która nie znajduje się w słowniku, należy do słownika wstawić frazę utworzoną przez konkatenację ostatnio wyprowadzonej frazy i pierwszego symbolu tej frazy oraz wyprowadzić tak utworzoną frazę na wyjście.

Tabela 4

Kodowanie LZW

Koder			Dekoder		
symbole pozostałe do zakodowania	kod	fraza wstawiana do słownika	kod	wyprowadzana fraza	fraza wstawiana do słownika
b|a|rbararabarbarbar	1	*ba* (3)	1	*b*	—
a|r|bararabarbarbar	0	*ar* (4)	0	*a*	*ba* (3)
r|b|ararabarbarbar	2	*rb* (5)	2	*r*	*ar* (4)
ba|r|arabarbarbar	3	*bar* (6)	3	*ba*	*rb* (5)
r|a|rabarbarbar	2	*ra* (7)	2	*r*	*bar* (6)
ar|a|barbarbar	4	*ara* (8)	4	*ar*	*ra* (7)
a|b|arbarbar	0	*ab* (9)	0	*a*	*ara* (8)
bar|b|arbar	6	*barb* (10)	6	*bar*	*ab* (9)
barb|a|r	10	*barba* (11)	10	*barb*	*barb* (10)
a|r|	4	—	4	*ar*	*barba* (11)

W kompresorach słownikowych stosuje się zazwyczaj słownik o określonej pojemności. Słownik taki w miarę postępowania kompresji wypełnia się nowymi frazami aż do całkowitego wypełnienia. Do momentu wypełnienia słownika algorytm jest adaptacyjny. W momencie gdy słownik jest pełny, można zaprzestać wstawiania nowych fraz, tj. „zamrozić słownik" [10]. W tym przypadku dla reszty pliku algorytm będzie stały. Można także powtórnie zainicjalizować słownik i budować go od nowa dla nowych danych. W algorytmie LZC [23], stosowanym przez kompresor compress zawarty w systemie Unix, łączy się oba rozwiązania. Po wypełnieniu słownika zaprzestaje się wstawiania nowych fraz i monitoruje się chwilowy współczynnik kompresji. W razie pogorszenia się jego wartości wykonuje się powtórną inicjalizację słownika.

W przeciwieństwie do występującego w algorytmie LZ77 słownika o postaci bufora z określoną liczbą ostatnio przetworzonych symboli komunikatu, słownik dla algorytmów z grupy LZ78 nie zawiera powtórzeń fraz. Z drugiej jednak strony w słowniku występują jedynie wyznaczone w procesie kodowania frazy, a nie wszystkie frazy, które można wyróżnić w już przetworzonej części komunikatu. Umożliwiające lepszą kompresję dłuższe frazy pojawiają się w słowniku stosunkowo powoli. W roku 1988 J.A. Storer zaproponował algorytm LZW-AP, w którym po zakodowaniu danej frazy słownik uzupełnia się o ciągi składające się z poprzedniej frazy i wszystkich przedrostków danej frazy, co prowadzi do szybszego wypełniania słownika. W roku 1989 E.R. Fiala i D.H. Greene przedstawili algorytm LZFG [2] będący hybrydą algorytmów LZ78 i LZ77. Wyznaczone w procesie kodowania frazy umieszczane są w słowniku dla algorytmu LZFG jako wskazania do bufora z ciągiem ostatnio przetworzonych symboli, dzięki czemu możliwe jest wyszukiwanie w słowniku fraz dłuższych, niż wcześniej wyznaczone.

7. Podsumowanie

W niniejszym artykule przedstawiono przegląd podstawowych pojęć, metod i algorytmów z zakresu bezstratnej kompresji danych. W artykule omówiono pojęcia podstawowe w bezstratnej kompresji danych, zarówno pojęcia z zakresu teorii informacji znajdujące zastosowanie w kompresji danych, jak i te dotyczące wyłącznie kompresji. Następnie przedstawiono podstawowe uniwersalne algorytmy kompresji, które, chociaż uznawane są już za klasyczne, są nadal powszechnie stosowane. Omówiono następujące algorytmy: algorytm Huffmana, algorytm kodowania arytmetycznego, algorytm kodowania długości sekwencji oraz algorytmy słownikowe LZ77, LZ78. Omówiono również istotne warianty ww. algorytmów oraz przedstawiono przykłady ilustrujące ich działanie. Praktycznie we wszystkich obecnie stosowanych algorytmach kompresji, w tym w algorytmach kompresji konkretnych klas danych, jak na przykład obrazy, w algorytmach kompresji stratnej oraz w najnowszych uniwersalnych algorytmach bezstratnej kompresji danych, występują omówione, podstawowe algorytmy. Algorytmy te występują w swojej podstawowej postaci lub jako wariant podstawowego algorytmu i zazwyczaj stanowią jeden z elementów większego i bardziej złożonego algorytmu. Algorytmy najnowsze są tematem oddzielnego opracowania, do którego artykuł niniejszy stanowi wprowadzenie [22].

Podziękowanie. Artykuł niniejszy powstał w wyniku projektu badawczego nr BW-486/RAu-2/2002 zrealizowanego w roku 2002 w Instytucie Informatyki Politechniki Śląskiej.

LITERATURA

1. Drozdek A.: Wprowadzenie do kompresji danych. WNT, Warszawa 1999.
2. Fiala E. R., Greene D. H.: Data Compression with Finite Windows. Communications of the ACM, Apr. 1989, Vol. 32(4), pp. 490-505.
3. Gallager R. G.: Variations on a theme of Huffman. IEEE Transactions on Information Theory, IT-24, 1978, pp. 668-74.
4. Huffman D. A.: A method for the construction of minimum-redundancy codes. Proceedings of the Institute of Radio Engineers 40(9), 1952, pp. 1098-101.
5. Howard P. G., Vitter J. S.: Analysis of arithmetic coding for data compression. Information Processing & Management, Great Britain 1992, Vol. 28(6), pp. 749-63.
6. Howard P. G., Vitter J. S.: Practical Implementations of Arithmetic Coding. rozdział w: Storer J. A., Image and text compression. Kluwer Academic Publishers 1992, pp. 85-112.

7. Howard P. G., Vitter J. S.: Arithmetic Coding for Data Compression. Proceedings of the IEEE, June 1994 USA, Vol. 82(6), pp. 857-65.
8. Knuth D. E.: Dynamic Huffman coding. Journal of Algorithms, Vol 6, 1985, pp. 163-80.
9. Lei S.-M.: On the finite-precision implementation of arithmetic codes. Journal of Visual Communication and Image Representation, March 1995 USA, Vol. 6(1), pp. 80-8.
10. Migas A.: Kompresja danych. Charakterystyka metody LZW. Zeszyty Naukowe Politechniki Śląskiej, Nr 1306, seria Informatyka z. 29, Wydawnictwo Politechniki Śląskiej, Gliwice 1995, ss. 95-116.
11. Moffat A., Neal R. M., Witten I. H.: Arithmetic Coding Revisited. ACM Transactions on Information Systems 1998, Vol. 16(3), pp. 256-94.
12. Rissanen J.: Universal Coding, Information, Prediction, and Estimation. IEEE Transactions on Information Theory, July 1984, Vol. 30(4), pp. 629-36.
13. Rissanen J., Landgon G. G.: Universal Modeling and Coding. IEEE Transactions on Information Theory, January 1981, Vol. 27(1), pp. 12-23.
14. Rissanen J., Landgon G. G.: A General Approach to Data Compression: Binary Arithmetic Codes. IBM Research Report RJ 7443, March 1990, IBM Almaden Research Center, San Jose, CA, USA.
15. Sayood K.: Kompresja danych wprowadzenie. RM, Warszawa 2002.
16. Shannon C. E.: A Mathematical Theory of Communication. Bell System Technical Journal, 1948, Vol. 27, pp. 379-423, 623-56.
17. Skarbek W.: Metody reprezentacji obrazów cyfrowych. Akademicka Oficyna Wydawnicza PLJ, Warszawa 1993.
18. Praca zbiorowa pod redakcją W. Skarbka: Multimedia algorytmy i standardy kompresji. Akademicka Oficyna Wydawnicza PLJ, Warszawa 1998.
19. Stuiver L., Moffat A.: Piecewise Integer Mapping for Arithmetic Coding. Proceedings of the IEEE Data Compression Conference (DCC'98), USA 1998, pp. 3-12.
20. Storer J. A., Szymański T. G.: Data compression via textual substitution. Journal of the Association for Computing Machinery, vol.29, no. 4, October 1982, USA, pp. 928-51.
21. Starosolski R.: Algorytmy bezstratnej kompresji obrazów. Studia Informatica, Vol. 23, Nr 4(51), Gliwice 2002, ss. 277-300.
22. Starosolski R.: Nowoczesne algorytmy bezstratnej kompresji danych. Studia Informatica, Vol. 24, Nr 1(52), Gliwice 2003, ss. 159-69.
23. Thomas S., Orost J.: Compress (version 4.0) program and documentation. 1985.
24. Vitter J. S.: Design and Analysis of Dynamic Huffman Codes. Journal of the ACM, Vol. 34(4), October 1987, pp. 825-45.
25. Vitter J. S.: Algorithm 673: Dynamic Huffman Coding. ACM transactions on Mathematical Software, Vol. 15(2), 1989, pp. 158-67.

26. Witten I. H., Moffat A., Bell T. C.: Managing Gigabytes. Van Nostrand Reinhold, 1994.
27. Ziv J., Lempel A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, Vol. 32(3), May 1977, pp. 337-43.
28. Ziv J., Lempel A.: Compression of individual sequences via variable rate coding. IEEE Transactions on Information Theory, Vol. 24(5), Sept. 1978, pp. 530-6.

Recenzent: Prof. dr hab. inż. Konrad Wojciechowski

Wpłynęło do Redakcji 11 grudnia 2002 r.

Abstract

In this paper, after presenting basic definitions and methods concerning lossless data compression, we describe basic universal lossless data compression algorithms that although regarded as classical, are still commonly used. We describe: Huffman Coding, Arithmetic Coding, Run Length Encoding and selected dictionary compression algorithms. We also describe important variants and present examples of the above-mentioned algorithms. Practically in every compression algorithm used presently, including compression algorithms designed for certain classes of data, like images, lossy compression algorithms and modern universal lossless data compression algorithms one or more of the described basic universal data compression algorithms is used. Those algorithms are used either in a basic form or as a variant of the basic form and usually are components of a substantially more complex algorithm.

Adres

Roman STAROSOLSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, rstaros@star.iinf.polsl.gliwice.pl .