ZESZYTY ETI ZESPOŁU SZKÓŁ W TARNOBRZEGU
Nr 1 Seria: Teleinformatyka 2013

Bogusław Rzeszut

Zespół Szkół im. ks. S. Staszica w Tarnobrzegu

KOMPRESJA BEZSTRATNA PLIKÓW – ALGORYTM HUFFMANA

Streszczenie

Referat zawiera szczegółowe omówienie pojęcia algorytmu Huffmana. W kilku przykładach pokazuje krok po kroku zasadę kompresji przy użyciu tej metody bezstratnej kompresji.

1. WSTĘP

Niniejszy artykuł zawiera przegląd podstawowych pojęć związanych ze sposobami kompresji danych. Dokładniej skupia się na sposobie kompresji za pomocą algorytmu Huffmana, opisuje go, określa najlepsze zastosowanie, porównuje z podobnymi sposobami kompresji, oraz przedstawia kilka prostych przykładów użycia algorytmu Huffmana.

2. PODSTAWOWE POJĘCIA DOT. KOMPRESJI

W znaczeniu informatycznym kompresja to zmniejszenie objętości danych przy zachowaniu „ładunku informacyjnego", czyli sensu tych danych. Celem kompresji jest zatem możliwie dokładna reprezentacja informacji przy użyciu możliwie małej ilości bitów. Kompresja ma zazwyczaj na celu oszczędność nośnika i/lub łącza sieciowego, którym są przesyłane dane, czyli redukcje kosztów. Proces odwrotny do kompresji nazywamy dekompresja. Kompresja jest możliwa, gdyż duża cześć danych cechuje się znaczna redundancja (nadmiarowością), tzn. Pewne informacje powtarzają się z różna częstością, dane są prezentowane w rozmaity sposób (np. grafika może być rastrowa lub wektorowa).

Kompresji poddawane są dane różnego rodzaju, mogą to być:

- Tekst
- Dźwięk (mowa, muzyka)
- Obraz ruchomy i nieruchomy
- Pliki wykonywalne

Zaletami kompresji danych są:

- przesyłanie większej ilości danych w jednostce czasu
- przesyłanie tej samej ilości danych w krótszym czasie
- zmniejszenie rozmiarów danych przechowywanych na nośnikach
- Praca większej liczby użytkowników na łączu o tej samej przepustowości (Internet)
- Duże rozmiary i upakowanie przechowywanych współcześnie danych
- Wygoda operowania plikami o mniejszych rozmiarach

Wady kompresji plików:

- Konieczność wykonania dekompresji zanim dane mogą być użyte dane użyte
- Czasami wymagana duża moc obliczeniowa, aby kompresja/dekompresja mogła być wykonywana w czasie rzeczywistym

Algorytmem kompresji nazywamy schemat postepowania przy zmniejszaniu objętości pliku; najczęściej zależy on od charakteru kompresowanych danych. Algorytmy kompresji można podzielić ze względu na różne kryteria, np. stopień zmiany „ładunku informacyjnego":

- kompresja bezstratna
- kompresja stratna

Algorytmy kompresji bezstratnej umożliwiają takie przechowanie danych, by w procesie dekompresji uzyskać dane w postaci identycznej z ta, jaka miały przed poddaniem ich kompresji. Dzieje się to jednak zazwyczaj kosztem gorszego współczynnika kompresji. Nadają się do danych charakteryzujących się duża redundancja (nadmiarowością) informacji. Stosuje się je głównie w tekstach, bazach danych, pewnych rodzajach obrazów statycznych (np. do zastosowań medycznych).
Przykłady algorytmów bezstratnych: Deflate, Huffman, LZW, RLE, BZIP2.

Współczynnik kompresji (ang. *compression ratio*) jest definiowany jako stosunek objętości danych skompresowanych (wyjściowych) do objętości danych oryginalnych (wejściowych), czyli jego wartość zawiera się w przedziale od 0 do 1.

$$WspKomp = \frac{dl(wej) - dl(wyj)}{dl(wej)} \cdot 100\%$$

Rys. 1 – Współczynnika kompresji - wzór [10]

Stopień kompresji (ang. *compression factor*) jest odwrotnością współczynnika kompresji i przyjmuje wartości większe od 1.

Algorytm zachłanny - algorytm, który w celu wyznaczenia rozwiązania w każdym kroku dokonuje zachłannego, tj. najlepiej rokującego w danym momencie wyboru rozwiązania częściowego. Innymi słowy algorytm zachłanny nie patrzy czy w kolejnych krokach jest sens wykonywać dane działanie, dokonuje on wyboru wydającego się w danej chwili najlepszym, kontynuując rozwiązanie kolejnego problemu wynikającego z podjętej wcześniej decyzji.

3. ALGORYTM HUFFMANA

3.1. Opis i podział Algorytmu Huffmana

Algorytm Huffmana, nazwany tak od nazwiska jego twórcy, został ogłoszony drukiem w 1952 roku, a więc ponad poł wieku temu. Mimo to pozostaje nadal jednym z najbardziej rozpowszechnionym algorytmem kompresji bezstratnej, dzięki efektywności, względnej prostocie i brakowi zastrzeżeń patentowych.

W słowniku kompresora Huffmana znajdują się wyłącznie elementy alfabetu wiadomości, tak więc rozmiar słownika jest stały i z góry znany (słownik nie jest rozbudowywany w czasie kompresji). Kody przydzielone poszczególnym znakom alfabetu są natomiast rożne. Kody dobiera się w taki sposób, żeby łączna długość skompresowanej wiadomości była jak najkrótsza. Osiąga się to wtedy, gdy długość kodu dla danego elementu alfabetu jest odwrotnie proporcjonalna do jego częstości występowania w wiadomości (najczęściej występujące elementy alfabetu mają najkrótsze kody).

Cechą charakterystyczną zestawu kodów Huffmana jest jego nieprefiksowość, oznacza to, że żaden kod Huffmana nie jest początkiem innego kodu. Znacznie upraszcza to budowę dekodera.

W idealnym przypadku częstości występowania poszczególnych znaków alfabetu wyznaczamy dokładnie, analizując całą wiadomość. W przypadku wiadomości bardzo długich, bądź też generowanych na bieżąco, jest to niemożliwe. Wtedy mamy do wyboru kilka rozwiązań:

1. Estymacja statyczna na podstawie początku wiadomości, lub wiedzy o procesie generującym wiadomość.
2. Estymacja blokowa, dzielimy wiadomość na duże bloki i dla każdego budujemy oddzielny słownik.
3. Estymacja adaptacyjna, statystyki są korygowane na bieżąco na bazie określonej ilości ostatnich znaków (tzw. okno kompresora).

3.1.1 Kody o stałej długości

Załóżmy, ze mamy do czynienia z plikiem który składa się z powtarzających pięciu znaków. Nazwijmy te znaki A, B, C, D, E. Do reprezentacji bitowej alfabetu pięcio- znakowego potrzebujemy jedynie (blg5c+1=3) trzech bitów. Wiec zapisując każdy z tych pięciu znaków odpowiednim dla niego nowym 3-bitowym kodem otrzymujemy plik stanowiący 37.5% pliku wejściowego. Takie kodowanie nazywamy kodowaniem o stałej długości, w sensie stałej długości nowych kodów przypisanych znakom.
Tabel ilustrująca problem kodowania o stałej długości:
plik: DABAADACDAC...

Znaki	A	B	C	D	E
Kody 8-bitowe	01010101	00001111	00110110	00011000	01010000
Kody 3-bitowe	000	001	010	011	100

3.1.2 Kody o zmiennej długości

Zauważmy, ze moglibyśmy zakodować jeszcze wydajniej plik gdybyśmy wiązali każdy znak z kompresowanego pliku nie z k-bitowym kodem, gdzie k= blgnc+1, gdzie n to mnogość alfabetu znaków dla pliku, tylko gdybyśmy każdy znak wiązali kolejno z jedno, dwu, trzy-bitowymi kodami aż do wyczerpania znaków. Dla poprzednio omawianego pliku kodowanie o zmiennej długości ilustruje następna tabela.
plik: DABAADACDAC...

Znaki	A	B	C	D	E
Kody 8-bitowe	01010101	00001111	00110110	00011000	01010000
Kody o zm. dłg.	0	1	00	01	10

Jednak stosując tak prosty dobór nowych kodów o zmiennej długości (tu zastosowaliśmy kolejne liczby binarne) powstaje problem z wieloznacznym odczytaniem zakodowanego pliku. Zauważmy, ze jeżeli wyżej kodowany plik (DABAADACDAC...) zapisany bitami po zakodowaniu go będzie wyglądał następująco (użyjmy spacji dla separowania kolejnych znakowi i lepszej czytelności): 01 0 1 0 0 01 0 00 01 0 00 ... , to nie jesteśmy w stanie odczytać go jednoznacznie - nie domyślimy się czy początkowe 010 to ABA, AE, czy DA. Co gorsza, nie jesteśmy w stanie rozwiązać problemu niejednoznacznego odczytania tak zakodowanego pliku. Zmuszeni wiec jesteśmy tak dobierać nowe wzorce bitowe aby odczytywanie pliku zapisanego tymi wzorcami było jednoznaczne.

3.1.3 Kody prefiksowe

Są one rozwiązaniem naszego nowego problemu doboru właściwych kodów o zmiennej długości. Kody prefiksowe są kadrami o zmiennej długości, a idea ich jest, aby kod znaku nie był prefiksem kodu innego znaku. W poprzednim doborze wzorców wzorzec dla A, czyli 0 był prefiksem dla 00 (C) i 01 (D). Dobierając wzorce zgodnie z zasada kodów prefiksowych musimy pamiętać, aby każdy nowo tworzony dla kolejnego znaku kod nie miał prefiksu będącego kodem wcześniej utworzonym dla innego znaku. Przykładowe właściwe przypisanie prefiksowych wzorców ilustruje następna tabela.
plik: DABAADACDAC...

Znaki	A	B	C	D	E
Kody 8-bitowe	01010101	00001111	00110110	00011000	01010000
Kody o zm. dłg.	0	10	110	1110	1111

Potrafimy wiec tworzyć poprawne kody prefiksowe. Zauważmy, ze najdłuższy kod z prefiksowych jest długości 4. Wiec jeżeli przykładowo nasz plik z pięcioma znakami miałby po 100 wystąpień znaku E, 20 wystąpień znaku D, po 10 wystąpień znaków C i D i 5 dla znaku A. Wtedy długość pliku kodowanego kodami prefiksowymi byłaby 100*3 + (20 + 10 + 10 + 5)*3 = 435 bitów, a długość pliku kodowanego wydajniejszymi kodami o zmiennej długości byłaby 100*4 + 5*1 + 10*2 + 10*3 + 20*4 = 535.

Aby kody o zmiennej długości (w szczególności prefiksowe) były wydajniejsze od kodów o stałej długości musimy znać częstość wystąpień każdego znaku w pliku, tak aby na przykład nie zdarzyło się, ze

najczęściej występujący znak ma najdłuższy kod. Będziemy przestrzegać zasady, aby kody przypisywać w kolejności ich długości, zaczynając od najkrótszego i przypisywać je znakom w kolejności ich częstości wystąpień w pliku, zaczynając od największej częstości. Ilustruje to następna tabela.
plik: DABAADACDAC...

Znaki	A	B	C	D	E
Częstość wyst.	5	10	10	20	100
Kody 8-bitowe	01010101	00001111	00110110	00011000	01010000
Kody o zm. dłg.	1111	1110	110	10	0

Obliczmy po wprowadzonych zmianach przypisania do znaków kodów długość pliku po kompresji. Wynosi ona 5*4 + 10*4 + 10*3 + 20*2 + 100*1 = 210. Potwierdza to przynajmniej dla naszego przykładu, ze kody o zmiennej długości są wydajniejsze od kodów o stałej długości. Faktycznie kody prefiksowe dają maksymalny stopień kompresji.

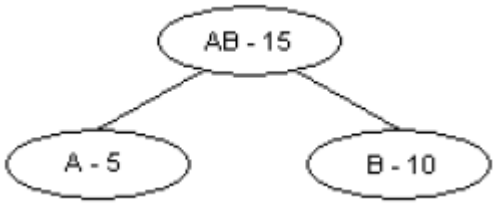
3.1.4 Drzewko binarne jako reprezentacja kodu prefiksowego

Kody prefiksowe dają maksymalny stopień kompresji, jak wcześniej wspomnieliśmy, ale na potrzeby wcześniejszych przykładów i rozważań nie generowaliśmy kodów prefiksowych w optymalny sposób, co jest kluczowe dla stopnia kompresji ta metoda. Do generowania optymalnego posłużą nam drzewa binarne, które są bardzo wygodna reprezentacja kodów prefiksowych. Metoda generowania kodów prefiksowych za pomocą drzew binarnych jest następująca:

1. Układamy nasz alfabet żaków dla kompresowanego pliku wg. kolejności częstości wystąpień znaku w pliku, zaczynając od najmniejszej częstości.
2. Pobieramy dwa pierwsze znaki z ułożonego alfabetu i tworzymy dla nich drzewo binarne w taki sposób, aby w węźle tego drzewa była suma częstości dla obu pogranych znaków, z znaki były liśćmi tego drzewa w kolejności: lewy liść - znak o mniejszej częstości, prawy - o większej.
3. Węzeł będący korzeniem nowo utworzonego drzewa dodajemy do alfabetu znaków jako kolejny znak ze swoja częstością. Powtarzamy procedurę aż do wyczerpania znaków w alfabecie - finalnie zostanie jeden znak.

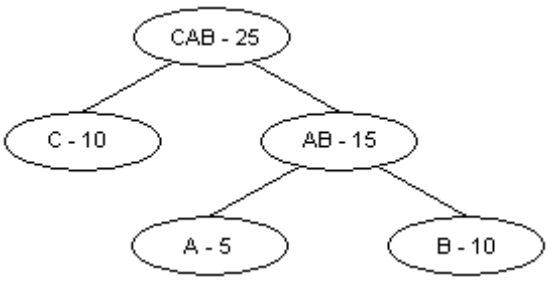
Dla pliku z poprzedniego przykładu ilustrują to kolejne kroki w procedurze:

1.1 uporządkowany alfabet znaków: A - 5, B - 10, C - 10, D - 20, E - 100
1.2 drzewo dla znaków A i B:



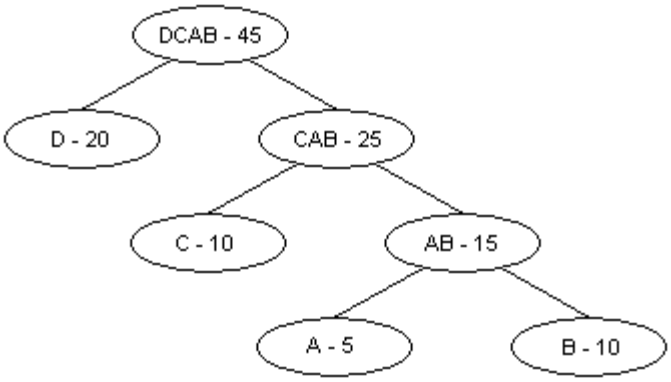
Rys. 2. Budowa drzewka algorytmu Huffmana – [1]

1.3 alfabet po dodaniu znaku AB: C - 10, D - 20, E - 100, AB - 15
2.1 uporządkowany alfabet znaków: C - 10, AB - 15, D - 10, E - 100
2.2 drzewo dla znaków C i AB:



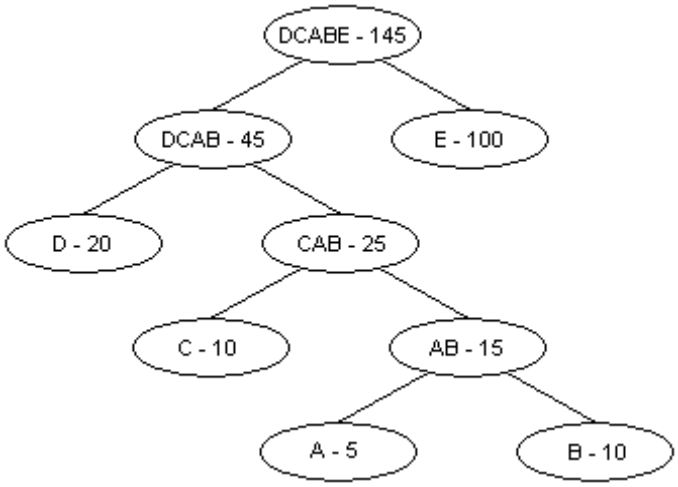
Rys. 3. Budowa drzewka algorytmu Huffmana – [1]

2.3 alfabet po dodaniu znaku CAB: D - 20, E - 100, CAB - 25
3.1 uporządkowany alfabet znaków: D - 20, CAB - 25, E – 100
3.2 drzewo dla znaków: D i CAB:



Rys. 4. Budowa drzewka algorytmu Huffmana – [1]

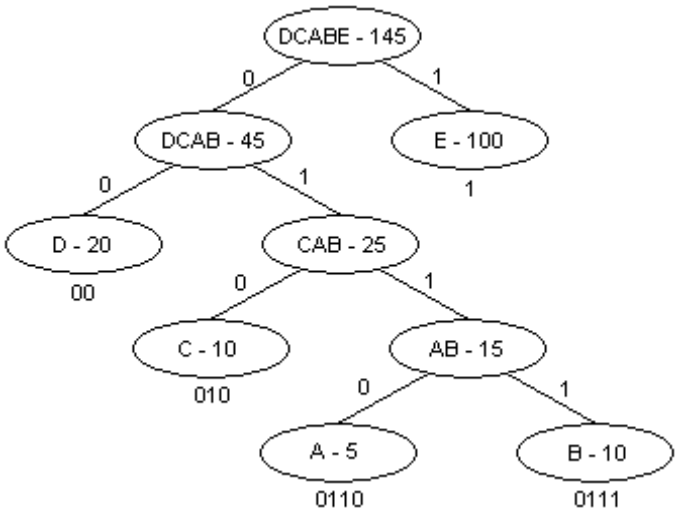
3.3 alfabet po dodaniu znaku DCAB: E - 100, DCAB - 45
4.1 uporządkowany alfabet znaków: DCAB - 45, E – 100
4.2 drzewo dla znaków: DCAB i E:



Rys. 5. Budowa drzewka algorytmu Huffmana – [1]

4.3 alfabet po dodaniu znaku DCABE: DCABE – 145

Do tak wygenerowanego drzewa binarnego każdej krawędzi łączącej węzeł z jego lewym synem przyporządkujmy wagę 0, a krawędzi łączącej węzeł z jego prawym synem przyporządkujmy wagę 1. Wtedy liście drzewa to pojedyncze znaki w pliku, a konkatenacja wag na drodze z korzenia do liścia to nowy prefiksowy kod binarny tego znaku w tym liściu. Ilustruje to następny rysunek.



Rys. 6. Budowa drzewka algorytmu Huffmana – [1]

3.1.5 Budowa algorytmu Huffmana

Jest to zachłanny algorytm obliczania kodów prefiksowych dla alfabetu znaków działający na zasadzie użyciem drzewa binarnego opisanej wcześniej.

```
Huffman(C)
1       n <- |C|
2       Q <- C
3       for i <- 1 to n - 1
4               do utwórz nowy węzeł w
5               lewy syn[w] <- x <- ExtractMin(Q)
6               prawy syn[w] <- y <- ExtractMin(Q)
7               częstość[w] = częstość[x] + częstość[y]
8               Insert(Q, w)
9 return ExtractMin(Q)
```

Algorytm Huffmana działa w zasadzie tak samo jak to robiliśmy wcześniej z użyciem drzewa binarnego. W pierwszym i drugim wierszu wartość n jest mnogością alfabetu dla naszego pliku, a Q jest kolejka znaków z alfabetu. Następnie algorytm buduje kolejne drzewa dla znaków o najmniejszej częstości wytapiania pobierając te znaki z kolejki Q metoda ExtractMin. Metoda insert jest potrzebna do wstawiania korzeni nowo powstałych drzew do kolejki Q zgodnie z porządkiem względem częstości wystąpień. Finalnie algorytm zwraca korzeń końcowego drzewa. Algorytm ten jest wydajna metoda obliczania kodów prefiksowych ponieważ jest on algorytmami zachłannym - wyborem zachłannym jest tu pobieranie z kolejki znaków znaku o najmniejszej częstości wystąpienia . Złożoność czasowa tego algorytmu zależy jednak bardzo od zastosowanych w nim metod ExtracMin i Insert i to one jako pod procedury algorytmu Huffmana decydują o jego końcowej złożoności czasowej. Dla przykładu: my generując drzewo binarne dla kodów za każdym wstawianiem do alfabetu nowego znaku (korzenia nowo utworzonego drzewa) sortowaliśmy alfabet. Jest to metoda intuicyjna choć naiwna i mało wydajna. Dużo szybsza metoda byłoby użycie kopców binarnych jako struktury danych przechowującej drzewa znaków. Wtedy inicjowanie kopca n znakami z alfabetu znaków (wiersz 2) przy użyciu procedury tworzenia kopca binarnego typu min kosztuje O(n). Pętla z wiersza 3 wykonuje n-1 iteracji, co przy procedurach pracy na kopcu ExtractMin i Insert daje złożoność O(n lg n). Cały koszt czasowy algorytmu Huffmana przy użyciu kopców binarnych jest O(n lg n).

3.2. Porównanie Algorytmu LZW i Huffmana

3.2. Algorytm LZW - informacje

Początki algorytmu LZW sięgają 1977 roku. Jego autorami są Abrahamm Lempel oraz Jacob Ziv, a w 1984 roku udoskonalił go Terry Welch (stad nazwa algorytmu - pierwsze litery nazwisk). Algorytm stosowany jest w systemie UNIX (funkcja compress) oraz w kompresji obrazów GIF (ang. graphics interchange format). Od

1995 roku chroniony jest patentem którego właścicielem jest firma Unisys Corporation. Jest to udoskonalona wersja kodowania metoda LZ78.

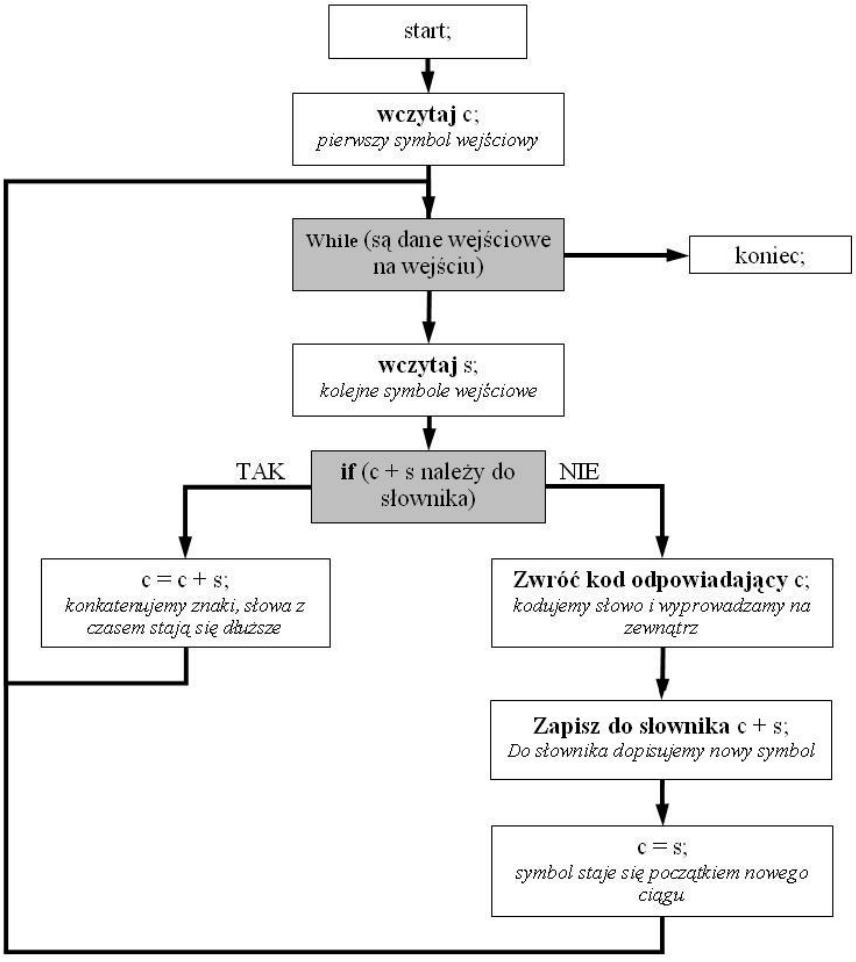
Algorytm LZW jest:

- algorytmem słownikowym, w którym nie istnieje predefiniowany słownik - jest on dynamicznie tworzony na podstawie ciągu danych wejściowych (podczas kodowania i dekodowania). Buduje swój słownik z występujących w danych wejściowych znaków.
- algorytmem bezstratnym - dane zdekodowane są identyczne z danymi przed kodowaniem.
- jednoprzebiegowy - dane są czytane i kodowane na bieżąco.
- jednoznaczny - odkodowanie słowa nie może prowadzić do dwóch różnych wyników, dwa różne słowa nie mogą mieć takiego samego kodu, co może prowadzić do nieścisłości. Jest to warunek konieczny kompresji.

Kodowanie metoda LZW

Metoda kodowania algorytmem LZW nie jest skomplikowana. Dane wejściowe są sprawdzane ze słownikiem, jeśli słowo występuje w słowniku to wczytujemy kolejny wyraz, leczymy z poprzednim i znów sprawdzamy, jeśli wyrazu nie ma w słowniku to dołączamy je do niego, nadajemy mu nowy kod a ostatni znak staje się pierwszym znakiem słowa nadchodzącego (nowego) i zwracamy kod ostatniego ciągu który był w słowniku.

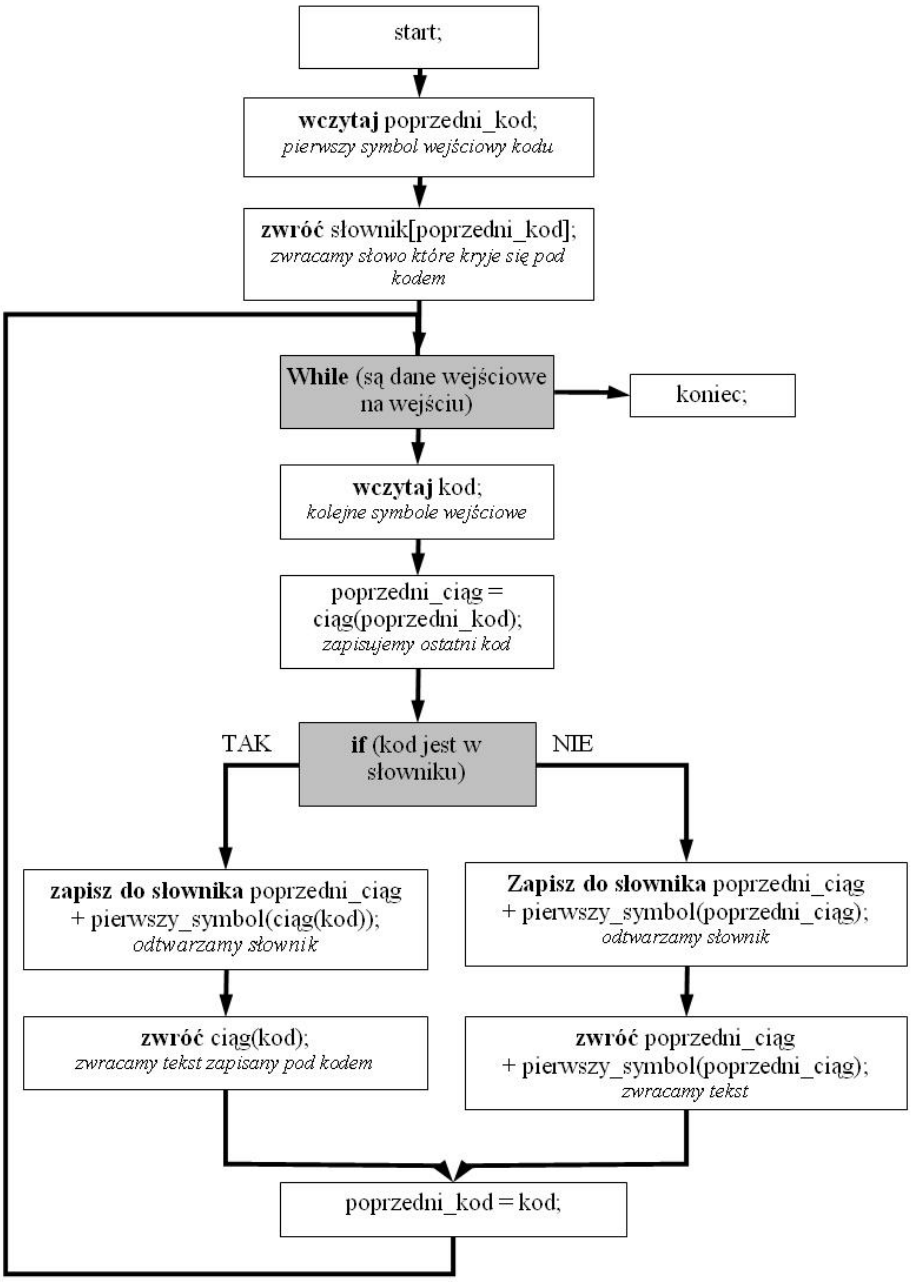
W algorytmie może ale nie musi być również wykorzystywany podstawowy i stały słownik (z pojedynczymi znakami). Jest to jednak rozwiązanie optymalizacyjne, taki słownik jest mały i bitowa reprezentacja kodów z takiego słownika będzie krótsza niż ze słownika dynamicznego.



Rys. 7. Kodowanie metoda LZW – [11]

Dekodowanie metoda LZW

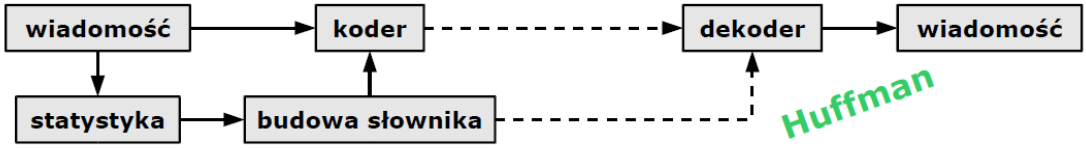
Dekodowanie jest procesem odwrotnym. Podczas wczytywania danych dekoder tworzy taki sam słownik jaki stworzył koder - co jest podstawa działania i warunkiem prawidłowej dekompresji. Słownik uaktualniany jest przy każdym przetwarzanym słowie kodu (oprócz pierwszego z nich). Dla każdego słowa kodu dekoder znajduje w słowniku i zwraca odpowiadający mu ciąg. Dekoder zapamiętuje ciąg i wczytywane jest następne słowo. Algorytm ten określa również działanie dla specjalnego przypadku, gdy słowo kodu nie ma w słowniku odpowiadającego mu hasła. Dzieje się tak gdy kodowany ciąg znaków zawiera podciąg "sCsCs", gdzie "s" jest pojedynczym symbolem, "C" jest ciągiem o dowolnej długości i "sC" znajduje się już w słowniku.



Rys. 8. Dekodowanie metoda LZW – [11]

3.2. Huffman a LZW

Nie należy uważać, że algorytm Huffmana jest zawsze lepszy od LZW, choć istotnie z reguły algorytm Huffmana daje lepszą kompresję (ale nie zawsze).

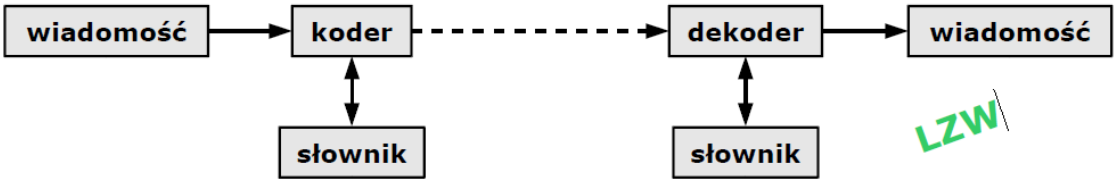


Rys. 9. Sposób działania algorytmu Huffmana – [2]

Należy pamiętać o dwoch potencjalnych wadach algorytmu Huffmana:

- Algorytm Huffmana nie posiada w słowniku sekwencji dłuższych niż 1 znak alfabetu.
- Algorytm Huffmana wymaga przesłania słownika do dekodera.

Pierwszą wadę łatwo sobie uzmysłowić, niech sekwencją do kompresji będzie ABCABCABCABC... Tu koder LZW szybko sobie w słownik wstawi symbol ABC (czy też BCA, CAB, obojętnie) i będzie emitował jeden symbol, co więcej wkrotce zbuduje sobie również dłuższe sekwencje typu ABCABC. Huffman będzie zaś bezradnie powtarzał trzy symbole na każde ABC... Druga wada jest oczywista – konieczność przesłania słownika zwiększa rozmiar kompresowanego pliku, co pogarsza ostateczną efektywność kompresji.

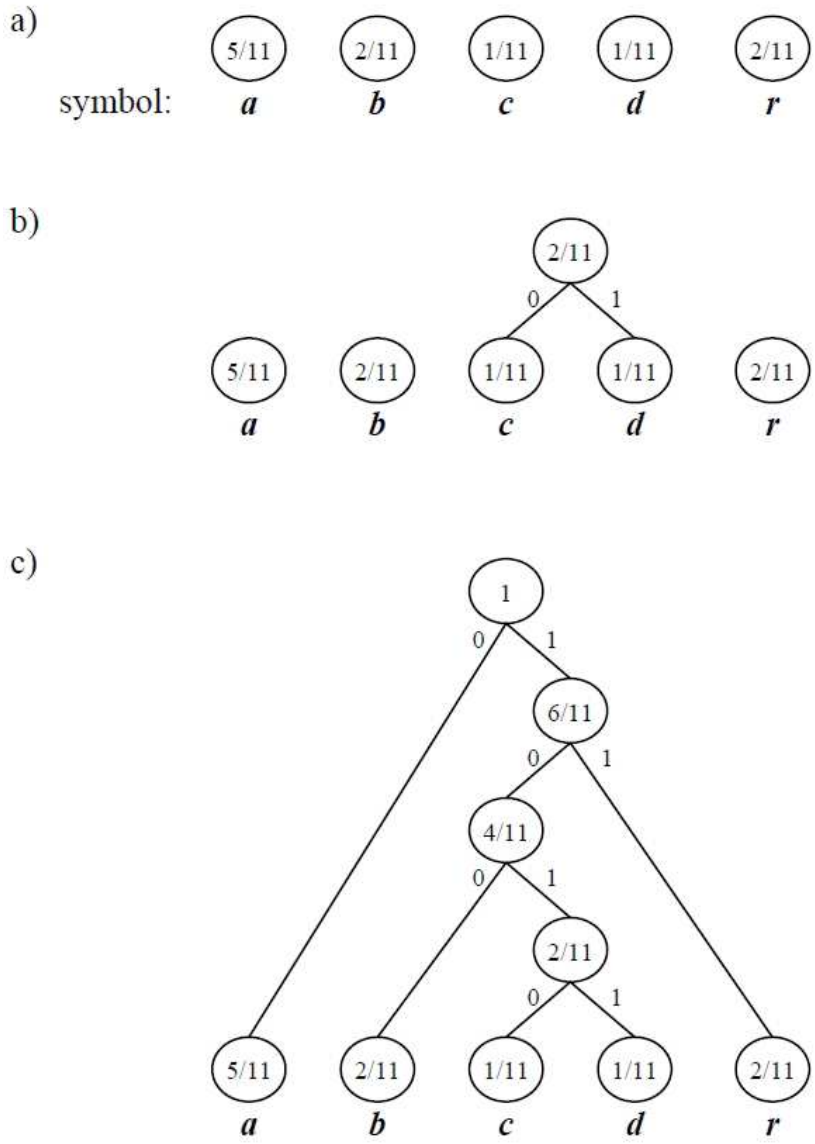


Rys. 10. Sposób działania algorytmu LZW – [2]

3.3. Dodatkowe przykłady kodowania algorytmem Huffmana

Przykład 1.

Budujemy drzewo Huffmana dla komunikatu ***abracadabra***. Alfabet źródła to {***a***, ***b***, ***c***, ***d***, ***r***}. W pierwszym kroku tworzonych jest 5 drzew (rys. 11.a). Wagi poszczególnych drzew to prawdopodobieństwa występowania odpowiadających im symboli wyznaczone na podstawie liczby wystąpień tych symboli w kodowanym komunikacie. Na rys. 11.b widnieje efekt pierwszego przeprowadzenia kroku 2. Rys. 11.c przedstawia gotowe drzewo Huffmana. Kod Huffmana dla tego drzewa umieszczono na rysunku 11.



Rys. 11. Budowanie drzewa Huffmana komunikatu **abracadabra** - [8]

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

Rys.12 Kod Huffmana dla drzewa z rys. 3.c - [8]

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 (rysunek 13), to efektywność kodu wzrośnie do około 72 %.

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

Rys.13 Kod Huffmana dla par symboli - [8]

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 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.

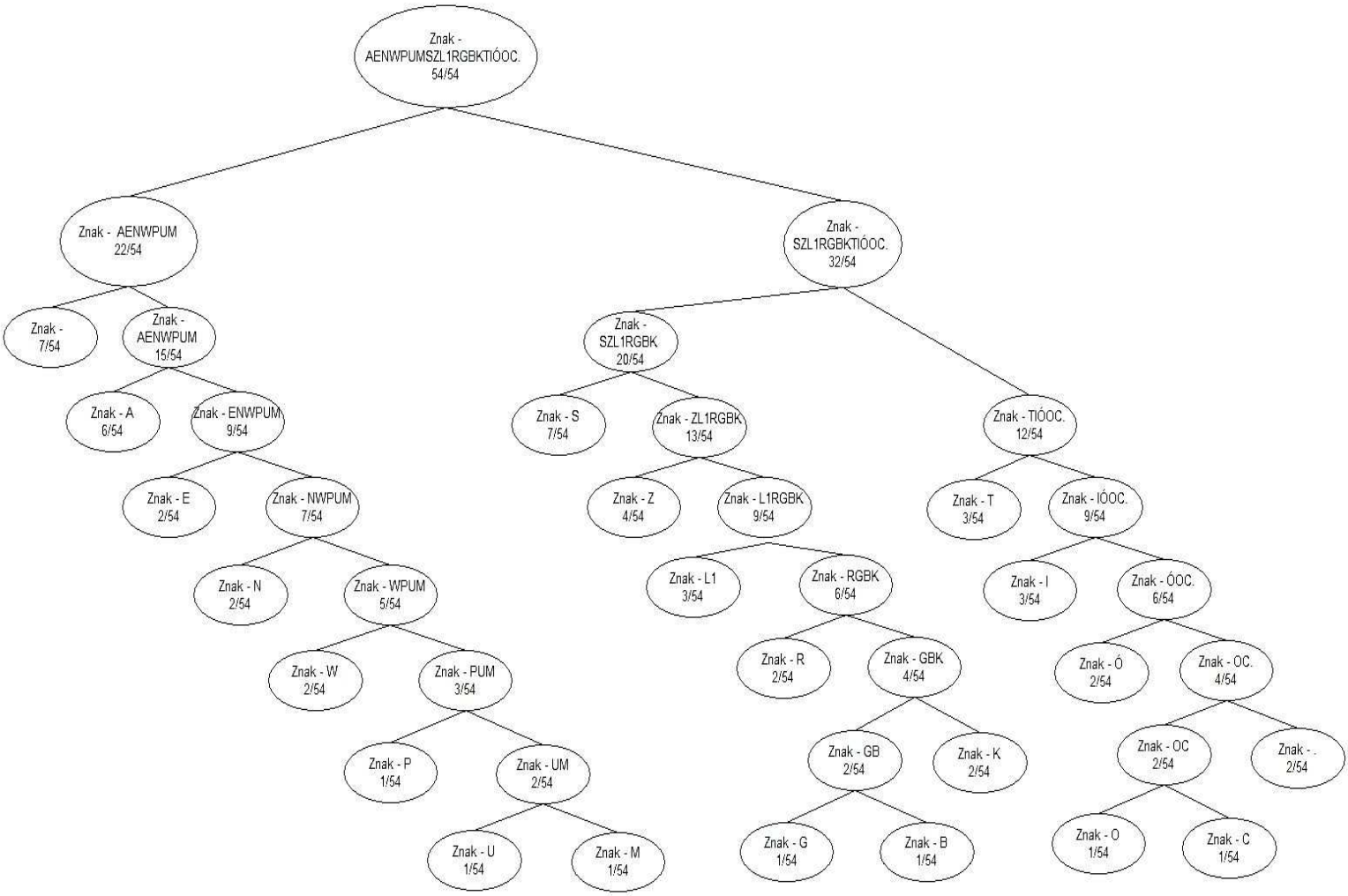
Przykład 3

Kodujemy ciąg 54 znaków stanowiących następujące zdanie: ZESPÓŁ SZKÓŁ IM. KS. STANISŁAWA STASZICA W TARNOBRZEGU
Następnie ustalamy częstotliwość występowania poszczególnych znaków w zdaniu, oraz analogicznie do poprzednich przykładów tworzymy drzewko dla algorytmu. W kolejnym kroku pojawia się porównanie kodu ASCII dla podanego zdania z tym samym zdaniem zakodowanym algorytmem Huffmana. Wyraźnie widać, że kompresja danych przyniosła oczekiwany elekt, ciąg bitów jest znacznie krótszy. Kod jest jednak nieefektywny w przypadku gdy prawdopodobieństwo wystąpienia danego znaku wynosi 1 do 54, gdyż zajmuje 16 bitów, identycznie jak w przypadku ASCII.

Występujące znaki, oraz częstotliwość ich występowania:

Znak	Częstotliwość występowania	Znak	Częstotliwość występowania
	7/54	K	2/54
S	7/54	.	2/54
A	6/54	W	2/54
Z	4/54	C	1/54
T	3/54	P	1/54
I	3/54	M	1/54
Ł	3/54	U	1/54
E	2/54	B	1/54
N	2/54	G	1/54
R	2/54	O	1/54
Ó	2/54		

Gotowe drzewko Huffmana:



Rys.14 Drzewko Huffmana dla przykładu 3

Zakodowane znaki:

Znak	Kod Huffmana	Kod ASCII	Znak	Kod Huffmana	Kod ASCII
	00	00100000	K	0111111	01001011
S	100	01010011	.	111111	00101110
A	010	01000001	W	011110	01010111
Z	1010	01011010	C	1111101	01000011
T	110	01010100	P	0111110	01010000
I	1110	01001001	M	01111111	01001101
Ł	10110	01001100	U	01111110	01010101
E	0110	01000101	B	01111101	01000010
N	01110	01001110	G	01111100	01000111
R	101110	01010010	O	1111100	01001111
Ó	11110	01001111			

Zdanie w postaci kodu ASCII:

0101101001000101010100110101000001001111010011000010000001010011010110100101011010011110100110000100000010010010100110100101110001000000100101101010011000010101011101000001001000000101001101010100010000010101001101011010010010100000110100000100100000010101110010000001010100010000010101001101011010010000111010000010010000001010111001000000101010001000001010100100100111001001111010000100101001001011010010001010100011101010101

Zdanie zakodowane z użyciem algorytmu Huffmana:

1010011010001111101011000100101001111111111010110001110011111111111111000111111001111110010011001001110111010010110010011111001000100110010100101011101111101010000111100011001010111001110111111000111111011011100110011111100011111110

W ustaleniach metody Huffmana należy założyć, że symbole wejściowe są 8bitowe. Wyznaczone słowa kodowe dla wszystkich symboli źródła powinny być zapisane w nagłówku zakodowanego pliku w postaci par: (symbol: kod) należy zaproponować format nagłówka.

Bezpośrednio za nagłówkiem rozpoczyna się zakodowana treść pliku. Należy pamiętać, że poszczególne słowa kodowe mają różną długość i powinny być wypisywane na wyjściu w postaci strumienia bitów (jedno za drugim, bez jakichkolwiek separatorów). Z uwagi na to, że kod Huffmana jest przedrostkowy, dekoder nie będzie miał problemów z jednoznacznym wykryciem początku każdego słowa kodowego.

Ciąg danych 01010011:100 stanowi informację o znaku (znak w kodzie ASCII: zakodowany znak) dla symbolu ‘S’.

Ciąg bitów poniżej zawiera nagłówek pozwalający odczytać sposób kodowania algorytmu dekoderom, oraz zakodowane zdanie ZESPÓŁ SZKÓŁ IM. KS. STANISŁAWA STASZICA W TARNOBRZEGU

00100000:00 01010011:100 01000001:010 01011010:1010 01010100:110 01001001:1110 01001100:10110 01000101:0110 01001110:01110 01010010:101110 01001111:11110 0111111: 01001011 111111: 00101110 011110: 01010111 1111101: 01000011 0111110: 01010000 01111111: 01001101 01111110: 01010101 01111101: 01000010 01111100: 01000111 1111100: 01001111

101001101000111110101100010010100111111111101011000111001111111111111000111111100111111001
001100100111011101001011001001111001000100110010100101011101111101010000111100011001010111
001110111110001111101101110011001111110001111110

Nawet wzrokowo widzimy, że kod Huffmana jest krótszy. Dokonaliśmy zatem kompresji danych. Kod Huffmana daje efektywną kompresję tylko wtedy, gdy częstości występowania symboli różnią się od siebie, a zakodowane dane są stosunkowo dużych rozmiarów. W przeciwnym wypadku powstaje kod o stałej liczbie bitów na symbol. Informacja o strukturze drzewka jest przesyłana wraz z zakodowaną informacją w sposób „uzgodniony" z dekoderem. Pozwala to odczytać skompresowany strumień danych.

BIBLIOGRAFIA:

[1] **Konrad Wypyski**: *Kody Huffmana*
[2] **mgr inż. Grzegorz Kraszewski**: *Systemu multimedialne*
[3] **Kinga Dziedzic, Agata Wajda, Agnieszka Gurgul**: *Kompresja statycznego obrazu cyfrowego*
[4] **Jan Wojciechowski**: *Kody Huffmana*, algorytmy.org
[5] **mgr Jerzy Wałaszek**: *Algorytm Huffmana z listą i drzewem*, I Liceum Ogólnokształcące im. Kazimierza Brodzińskiego w Tarnowie
[6] **Tomasz Lewicki**: *Kompresja danych i formaty plików graficznych*, WWSIS Wrocław - 2007
[7] **Roman Starosolski**: *Algorytmy bezstratnej kompresji obrazów*, Politechnika śląska, instytut informatyki - 2002
[8] **Roman Starosolski**: *Algorytmy bezstratnej kompresji danych*, Politechnika śląska, instytut informatyki - 2003
[9] **Roman Starosolski**: *Algorytmy kodowania Shannona-fano i Huffmana*, Algorytmy kompresji danych – wykład 3
[10] **Radosław Kaczyński**: *Kompresja bezstratna*, Politechnika Wrocławska, Wrocław 2006
[11] **Konrad Włodarczyk**: *Kompresja i dekompresja danych algorytmem LZW*, Politechnika Warszawska wydział elektroniki i technik informacyjnych

LOSSLESS FILE COMPRESSION - HUFFMAN CODING

Summary

The paper contains a detailed discussion of the concept of the Huffman coding. Several examples show you step-by-step principle of compressed using the lossless compression methods.