Uniwersytet Zielonogórski
Instytut Sterowania i Systemów Informatycznych

Algorytmy i struktury danych

Laboratorium Nr 9

Kompresja

1 Cel ćwiczenia

Ćwiczenie ma na celu zapoznanie studentów z wybranymi zagadnieniami dotyczącymi kompresji danych.

2 Wstęp

W informatyce występuje wiele zastosowań, w których stosujemy kompresję danych. Mimo ciągle powiększających się dysków twardych i pamięci RAM, nadal występuje potrzeba stosowania różnego rodzaju kompresji danych.

3 Kodowanie

Kodowaniem nazywa się takie przyporządkowanie, że każdemu symbolowi ze zbioru ω przyporządkowuje się kod $k(\omega)$, np.

$$\omega = \{a_1, a_2, a_3, \ldots, a_n\}$$
$$k(\omega) = \{k(a_1), k(a_2), k(a_3), \ldots, k(a_n)\}$$

gdzie

$$k(a_1), k(a_2), k(a_3), \ldots, k(a_n) \in (0,1)^n.$$

Pojedyncze słowo kodowe a_i kodowane jest następująco

$$\omega = a_{i1}, a_{i2}, a_{i3}, \ldots, a_{in} \quad \rightarrow \quad k(\omega) = \{k(a_{i1}), k(a_{i2}), k(a_{i3}), \ldots, k(a_{in})\}$$

4 Entropia

Proces kompresji danych jest odwracalnym procesem wtórnego kodowania danych, realizowanych w celu zmniejszenia ich redundancji.

Szybkość przekazu przesyłanej informacji zależy od wyboru reprezentacji danych. Poniżej przedstawione zostaną różne metody kompresji danych zmniejszające rozmiar reprezentacji bez wpływu na treść informacji. Najpierw jednak zdefiniujemy pojęcie entropii.

Załóżmy, że do kodowania wiadomości używamy s niezależnych od siebie symboli o znanych prawdopodobieństwach $P(s_i)$. Symbole s tworzące zbiór S są kodowane za pomocą ciągów zer i jedynek. Wówczas prawdziwy jest następujący wzór

$$P(s_1) + P(s_2) + \cdots + P(s_n) = \sum_{i=1}^{n} P(s_i) = 1.$$

Informację zawartą w zbiorze S, zwaną entropią źródła S, definiuje się jako

$$H = P(s_1)L(s_1) + P(s_2)L(s_2) + \cdots + P(s_n)L(s_n) = \sum_{i=1}^{n} P(s_i)L(s_i), \tag{1}$$

gdzie

$$L(s_i) = \frac{1}{\log_2(P(s_i))} = -\log_2\big(P(s_i)\big), \quad i = 1, 2, \dots n$$

co stanowi minimalną długość kodu dla symbolu s_i. Liczba $L(s_i)$ wyraża jaka jest potrzebna minimalna ilość bitów, by symbol s_i mógł być w pełni zakodowany. Wzór na entropię H podał Claude E. Shannon.

Znając symbole tworzące kod oraz częstość ich występowania uzyskujemy z równania (1) najlepszą możliwą średnią długość kodu potrzebną do zakodowania tych symboli. Żaden algorytm kompresji danych nie może dawać wyniku lepszego niż H, a im bliższy jest tej liczby, tym lepszy jest jego współczynnik kompresji.

Przykładowo, gdy dane są trzy symbole s_1, s_2 i s_3 z prawdopodobieństwem odpowiednio $0.25, 0.25$ i 0.5, to długości (w bitach) przypisanych im kodów będą wynosiły

$$-\log_2(P(s_1)) = -\log_2(P(s_2)) = -\log_2(0.25) = \log_2\left(\frac{1}{0.25}\right) = \log_2(4) = 2$$

oraz

$$-\log_2(P(s_3)) = -\log_2(0.5) = \log_2\left(\frac{1}{0.5}\right) = \log_2(2) = 1$$

a średnia długość kodu będzie wynosiła

$$H = P(s_1)L(s_1) + P(s_2)L(s_2) + P(s_3)L(s_3) = 2P(s_1) + 2P(s_2) + P(s_3) = 0.5 + 0.5 + 0.5 = 1.5$$

W kodowaniu entropowym stosujemy słowa o zmiennej liczbie bitów. Miara optymalności kodu jest wyznaczana przez entropię. Jeżeli miara optymalności kodu jest średnią długością kodu, taką że

$$\bar{l} = \sum_{i=1}^{n} p_i l_i,$$

gdzie l_i – to minimalna długość kodu, p_i – prawdopodobieństwo jego wystąpienia, $\forall\ i = 1, 2, \dots, n$, to musi zachodzić następująca nierówność

$$\bar{l} \geqslant H.$$

Entropia jest więc dolną granicą kompresji oraz miarą wartości oczekiwanej z tytułu kompresji. Kod, którego długość jest równa entropii, jest kodem optymalnym. Konstruując kod optymalny minimalizujemy średnią długość kodu, zależną od prawdopodobieństwa P wystąpienia symbolu. Gdy symbol jest rzadko używany, to przypisuje mu się długi ciąg, a symbolowi często występującemu krótki ciąg. Kodowanie entropowe pozwala na osiągnięcie większych wartości stosunku kompresji wtedy, gdy wartości prawdopodobieństwa występowania poszczególnych symboli są bardziej zróżnicowane, co odpowiada zróżnicowaniu wartości histogramu obrazu. Jednakże, obraz o prawidłowym kontraście posiada histogram stosunkowo wyrównany, dlatego też kodowanie entropowe stosowane bezpośrednio do zakodowania obrazu pozwala na osiągnięcie współczynników kompresji rzadko przekraczających 1.5 i rzadko stosuje się je jako samodzielną metodę kompresji. Natomiast powszechnie stosuje się kodowanie entropowe do redundancji danych statystycznych.

5 Kody bezpośrednie

Dla jednoznacznego dekodowania zakodowanego ciągu symboli zdefiniowano tzw. *kody bezpośrednie*.

Założenia kodowania bezpośredniego:

- każdy kod odpowiada dokładnie jednemu symbolowi;
- kod posiada własność prefiksową, czyli kod żadnego symbolu nie jest prefiksem innego - kodowanie jest jednoznaczne;
- długość kodu danego symbolu nie powinna przekroczyć długości kodu symbolu mniej prawdopodobnego

  $$P(a_i) \geqslant P(a_j)$$
  $$L(a_i) \leqslant L(a_j)$$

  $\forall\ i \geqslant 1$ oraz $j \leqslant n$;
- nie powinno być żadnych nie wykorzystanych ciągów kodowych, jeżeli były kody dłuższe (o ile niewykorzystane kody nie są prefiksem żadnego innego).

5.1 Tworzenie kodów bezpośrednich

Tworzenie kodów bezpośrednich odbywa się na takiej zasadzie, aby wyeliminować złe ciągi kodowe o danej długości, tzn. będące prefiksami innych kodów. Jeżeli symbolom $a_1, a_2, \ldots, a_n$ odpowiadają ciągi kodowe o długości odpowiednio $d_1, d_2, \ldots, d_n$, takie że

$$d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n.$$

Jeżeli $k(a_1) = a$ o długości d_1 to złych ciągów o długości d_2 (czyli z prefiksem "a") jest

$$2^{d_2 - d_1}$$

$k(a_2)$ o długości d_2 da się ułożyć o ile

$$\begin{aligned} 2^{d_2} &> 2^{d_2 - d_1} \\ 2^{d_2} &\geqslant 2^{d_2 - d_1} + 1 \\ 1 &\geqslant 2^{-d_1} + 2^{-d_2} \end{aligned}$$

Dla następujących symboli można określić słowo kodowe z nierówności Krafta

$$1 \geqslant 2^{-d_1} + 2^{-d_2} + \cdots + 2^{-d_n}.$$

W kodowaniu bezpośrednim zakłada się, że prawdopodobieństwo wystąpienia symbolu a_i na pozycji i jest niezależne od i oraz pozostałych symboli, ponieważ udowodniono, że dla takiego kodu średnia długość kodu jest najmniejsza.

Przykład: Niech symbolom a, b, c i d odpowiadają odpowiednie słowa kodowe $01, 10, 001$ i 011. Następnie zakodowano pewną sekwencję i kod tej sekwencji ma następującą postać

$$0110010011001.$$

Dekodując dany ciąg bitów, począwszy od lewej mamy

$$abacba \qquad \text{lub} \qquad dccba.$$

Jak widać na powyższym przykładzie, przykład ten nie jest przykładem kodu bezpośredniego, ponieważ dekodowanie nie daje jednoznacznego wyniku. Można także pokazać, że słowo kodowe odpowiadające symbolowi a jest prefiksem dla d, czyli kod ten nie posiada własności prefiksowej.

6 Współczynnik kompresji

Współczynnik kompresji jest miarą służącą do porównania różnych metod kompresji (kodowania) i określony jest wzorem

$$\frac{D_{we} - D_{wy}}{D_{we}},$$

gdzie D_{we} – długość ciągu wejściowego, D_{wy} – długość ciągu wyjściowego. Współczynnik ten wyrażony w procentach informuje o ilości nadmiarowej informacji usuniętej z danych wejściowych.

7 Metoda Huffmana

Jest to metoda kodowania opracowana przez Davida Huffmana, wykorzystująca strukturę drzewa. Drzewo, wykorzystywane do kodowania i dekodowania symboli zakodowanych metodą Huffmana, tworzone jest przy pomocy procedury pokazanej w Tab. 7.1.

Oznaczenia przyjęte w tej procedurze są następujące: n – liczba kodowanych/dekodowanych symboli; L – liście drzewa (zobacz Rys. 7.1 Krok 0); P – prawdopodobieństwa dla poszczególnych symboli; Q – kolejka priorytetowa z operacją minimum względem wartości P (w tej kolejce są przechowywane wierzchołki drzewa, które nie zostały jeszcze przetworzone).

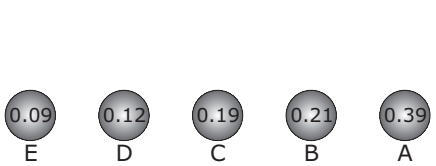
Algorytm Huffmana przedstawiony w Tab. 7.1 rozpoczyna swoją pracę od wyznaczenia ile elementów będzie przetwarzanych (n), a następnie wpisania liści drzewa (L) do kolejki priorytetowej Q z operacją minimum względem prawdopodobieństw przechowywanych w zmiennej P dla poszczególnych symboli. Dalej rozpoczyna się pętla główna procedury, która jest przetwarzana $n - 1$ razy. Najpierw tworzony

```
     Procedura Huffman(L)
1.   n ← |L|
2.   Q ← L
3.   Dla i ← 1 do n − 1 wykonaj
4.       z ← UtwórzNowyWęzełDrzewa
5.       x ← WybierzMin(Q)
6.       LewePodDrzewo[z] ← x          ▷ Waga dla krawędzi (z, x) = 0.
7.       y ← WybierzMin(Q)
8.       PrawePodDrzewo[z] ← y         ▷ Waga dla krawędzi (z, y) = 1.
9.       P[z] ← P[x] + P[y]
10.      WstawDoKolejki(Q, z)
11.  return WybierzMin(Q)              ▷ Wynikiem procedury jest korzeń drzewa.
```

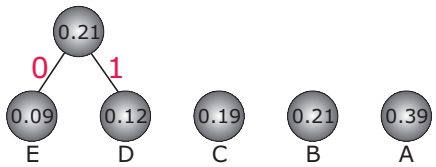
Tab. 7.1: Procedura Huffman.

jest nowy węzeł drzewa z (linia 4). Dalej wybierany jest minimalny węzeł z kolejki Q i wpisywany do zmiennej x. W linii 6 węzłowi z przypisujemy lewego syna w postaci węzła x (poruszając się po drzewie w lewo, krawędzi (z, x) jako lewej odnodze węzła z nadajemy wartość 0). Następnie znów wykonujemy to samo dla następnego elementu pobranego z kolejki Q. W linii 8 węzłowi z przypisujemy prawego syna w postaci węzła y (poruszając się po drzewie w prawo, krawędzi (z, y) jako prawej odnodze węzła z nadajemy wartość 1). Następnie obliczana jest nowa wartość prawdopodobieństwa dla nowego węzła z w postaci sumy prawdopodobieństw $P[z] = P[x] + P[y]$. W kolejnym i ostatnim już kroku pętli głównej do kolejki Q wstawiany jest nowo utworzony węzeł z. Cała ta procedura powtarza się $n-1$ raz. Na koniec w kolejce Q pozostanie jeden wierzchołek (korzeń drzewa), który zostaje pobrany z tej kolejki (linia 11) i zwrócony jako wynik działania procedury.

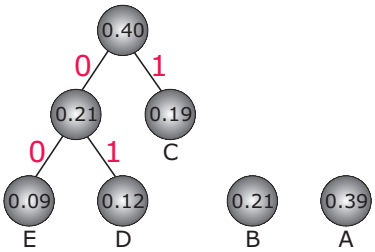
Przykład: Pięć liter: A, B, C, D, i E ma prawdopodobieństwo wystąpienia (w jakimś przykładowym



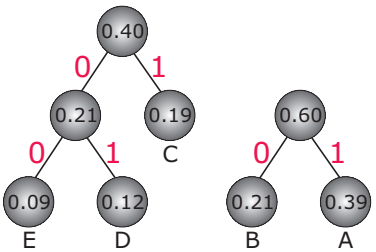
Krok 0



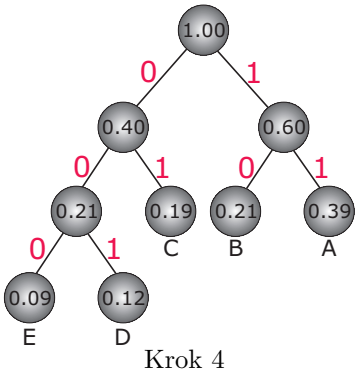
Krok 1



Krok 2



Krok 3



Krok 4

Symbol	Prawdopodobieństwo wystąpienia	Kod symbolu
A	0.39	11
B	0.21	10
C	0.19	01
D	0.12	001
E	0.09	000

Tabela kodu dla symboli

Rys. 7.1: Ilustracja budowy drzewa Huffmana (przypadek 1).

tekście) odpowiednio: 0.39, 0.21, 0.19, 0.12, 0.09.

Kod bitowy dla danego symbolu wyznacza się poprzez przejście drogi od korzenia do danego symbolu. Przechodząc tę drogę odczytujemy wartość 0 lub 1 na krawędziach prowadzących od korzenia do danego symbolu, którego kod chcemy odczytać (wartości te są przy krawędziach oznaczone kolorem czerwonym na Rys. 7.1 i 7.2).

Na wspomnianych rysunkach przedstawiono dwa przypadki budowy kodu Huffmana. Biorą się one stąd, że symbole mogą być grupowane w nieco innej kolejności, gdy mamy te same wartości prawdopodobieństwa. Rozważmy Rys. 7.1 i 7.2 w kroku 1. Możemy zauważyć, że wartość prawdopodobieństwa o wartości 0.21 występuje dwukrotnie. W związku z tym, w następnym kroku algorytmu (Krok 2) pobrany zostanie z kolejki Q węzeł odpowiadający symbolowi C o wartości 0.19 i wpisany pod zmienną x, a dalej pod zmienną y może zostać wpisany symbol, który powstał z połączenia symboli E i D o warości 0.21 lub symbol B o tej samej wartości. Ponieważ metoda Huffmana nie narzuca kolejności wybierania elementów, więc wszystko zależy od sposobu implementacji kolejki priorytetowej Q, w której może być najpierw symbol ED a potem B lub dokładnie odwrotnie.

Sekwencję BCAD z powyższego przykładu można zatem zapisać na dwa sposoby:

$$100111001 \qquad \text{lub} \qquad 111001001.$$

Dekodowanie w metodzie Huffmana odbywa się w następujący sposób:

1. Bity czytane są od lewej do prawej, aż do momentu wykrycia kodu pojedynczego symbolu określonego przez drzewo kodowe.
2. Bity te zostają usunięte z ciągu bitów, a zdekodowany symbol zapamiętany. Następuje ponowne wykonanie punktu 1.
3. Punkty 1. i 2. są wykonywane aż do momentu odczytania całego ciągu bitów.

Takie dekodowanie jest możliwe dzięki posiadaniu przez kod metody Huffmana własności prefiksowej.

Rozważmy dekodowanie dla drugiego przypadku tzn. mamy następujący zakodowany tekst:

111001001

Procedurę dekodowania zaczynamy od wierzchołka drzewa pokazanego na Rys. 7.2 w kroku 4. Poruszamy się od korzenia i idziemy najpierw w prawo, gdyż pierwszym bitem dla naszego przykładu jest 1 (111001001). Następnie idziemy znów w prawo bo mamy bit o wartości 1 (111001001). Dalej już iść nie możemy, gdyż jesteśmy na końcu drzewa – dotarliśmy do symbolu B o kodzie 11. Teraz usuwamy pierwsze dwa bity z zakodowanego ciągu i otrzymujemy

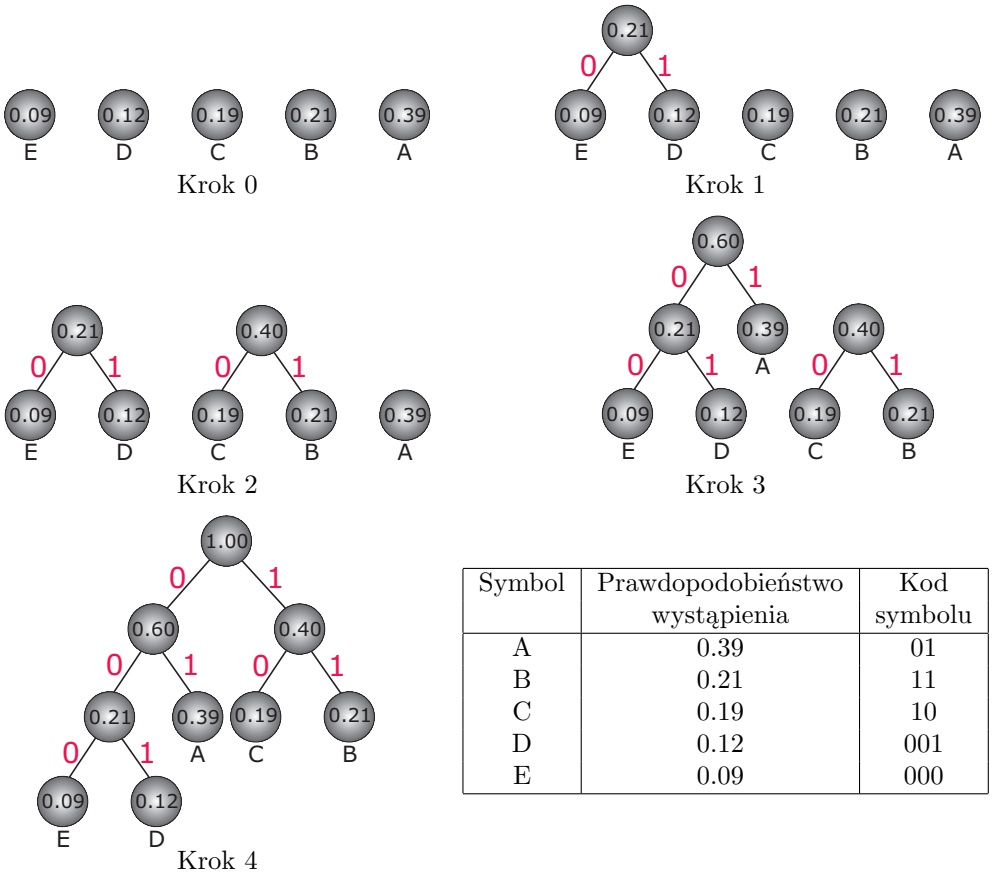
Zdekodowany symbol	O wartości	Pozostały zakodowany tekst
11	B	1001001

Następnie procedura rozpoczyna się od początku. Zaczynamy od korzenia idąc w prawo (bo mamy bit o wartości 1 – 111001001) , następnie w lewo (bo mamy bit o wartości 0 – 111001001) i nasza wędrówka dobiegła już końca. Otrzymaliśmy symbol C o kodzie 10.

Zdekodowany symbol	O wartości	Pozostały zakodowany tekst
11	B	1001001
10	C	01001

Następnie procedura rozpoczyna się od początku. Zaczynamy od korzenia idąc w lewo (bo mamy bit o wartości 0 – 111001001), następnie w prawo (bo mamy bit o wartości 1 – 111001001) i nasza wędrówka dobiegła już końca. Otrzymaliśmy symbol A o kodzie 01.

Zdekodowany symbol	O wartości	Pozostały zakodowany tekst
11	B	1001001
10	C	01001
01	A	001



Symbol	Prawdopodobieństwo wystąpienia	Kod symbolu
A	0.39	01
B	0.21	11
C	0.19	10
D	0.12	001
E	0.09	000

Rys. 7.2: Ilustracja budowy drzewa Huffmana (przypadek 2).

W ostatnim kroku postępujemy podobnie, więc najpierw w lewo (od korzenia – 1110010**01**), potem znów w lewo (111001**001**) i na koniec w prawo (111001**001**). Otrzymaliśmy symbol D o kodzie 001 – co kończy odkodowywanie naszego tekstu.

Kolor brązowy bitów oznacza, że bity te są już usunięte z początkowego zakodowanego kodu. Kolor czerwony bitów pokazuje którędy poruszamy się po drzewie Huffmana.

Zdekodowany symbol	O wartości	Pozostały zakodowany tekst
11	B	1001001
10	C	01001
01	A	001
001	D	

Do oszacowania efektywności metody Huffmana wykorzystuje się pojęcie ważonej ścieżki $L(m_i)$ interpretowanej jako liczba zer i jedynek w kodzie przypisanym symbolowi m_i. Korzystając z definicji średniej długości kodu można obliczyć średnią długość kodu ze wzoru (1) i wynosi ona

$$H = \sum_{i=1}^{5} P(s_i)L(s_i) = 0.09 \cdot 3.4739 + 0.12 \cdot 3.0589 + 0.19 \cdot 2.3959 + 0.21 \cdot 2.2515 + 0.3900 \cdot 1.3585 = 2.1376.$$

Dla porównania, wyliczmy ważoną długość ścieżki dla kodu Huffmana

$$H_{Huf} = \sum_{i=1}^{5} P(m_i)L(m_i) = 0.09 \cdot 3 + 0.12 \cdot 3 + 0.19 \cdot 2 + 0.21 \cdot 2 + 0.39 \cdot 2 = 2.21$$

Jak można zobaczyć kod Huffmana jest bardzo bliski entropii wyliczonej ze wzoru (1) i jest gorszy zaledwie o 3.28%.

Częstości występowania poszczególnych symboli nie zawsze są proste do wyznaczenia, dlatego w praktyce często stosuje się pewną odmianę metody Huffmana. Jest nią tzw. kompresja adaptacyjna. Są dwa typy rozwiązań takiej metody:

1. Użycie początkowego zbioru częstości występowania symboli powstałej na podstawie fragmentu przesyłanego tekstu lub jakieś hipotezy. Następnie jest budowana tabela konwersji, zawierająca dodatkowe pole z licznikiem, a sama tabela jest uaktualniana w trakcie transmisji danych.

2. Wersja czysto adaptacyjna, bez żadnych założeń odnośnie początkowego rozkładu symboli występujących w kodowanej wiadomości. Metodę taką opracował Robert G. Gallager. Została ona ulepszona później przez: Donalda Knuth i Jefrey'a S. Vitter'a.

8 Metoda Shannona-Fano

Efektywną metodą generowania kodu optymalnego przy n dążącym do nieskończoności (liczba kodowanych symboli jest bardzo duża – dąży do nieskończoności) jest metoda Shannona-Fano pokazana w Tab. 8.1.

Oznaczenia przyjęte w tym algorytmie: S – zbiór kodowanych symboli; n – liczba kodowanych symboli; S_1 i S_2 – podzbiory zbioru S. Zapis w linii 5: $|P(S_1) - P(S_2)| \rightarrow \min$, oznacza: znajdź minimalną różnicę prawdopodobieństw między zbiorami S_1 i S_2, taką, że jest ona minimalna (ze wszystkich innych możliwych podziałów na podzbiory dwuelementowe).

Twierdzenie Shannona:

$$L_{min}(s) \leqslant H(s) + 1 \tag{2}$$

gdzie $L_{min}(s)$ jest średnią długością najlepszego kodowania.

Przykład: Pięć liter: A, B, C, D, i E ma prawdopodobieństwo wystąpienia (w jakimś przykładowym tekście) odpowiednio: 0.39, 0.21, 0.19, 0.12, 0.09.

Najpierw ciąg $S = (A, B, C, D, E)$ dzielimy na podciągi $S_1 = (C, D, E)$ i $S_2 = (A, B)$, ponieważ różnica między $P(S_1) = P(C)+P(D)+P(E)$ a $P(S_2) = P(A)+P(B)$ jest najmniejsza spośród wszystkich możliwych podziałów S na dwa podciągi. Kod każdej litery z S_1 będzie zaczynał się od zera a kod liter z S_2 od jedynki.

S_1	Prawdopodobieństwo	Kod	S_2	Prawdopodobieństwo	Kod	Różnica
C	0.19	0	A	0.39	1	
D	0.12	0	B	0.21	1	
E	0.09	0				
Suma:	0.40			0.60		0.20

Następnie ciąg S_1 dzielimy na $S_{11} = (D, E)$ oraz $S_{12} = (C)$ i rozszerzamy kody dopisując dla D i E zero a dla C jedynkę.

S_{11}	Prawdopodobieństwo	Kod	S_{12}	Prawdopodobieństwo	Kod	Różnica
D	0.12	00	C	0.19	01	
E	0.09	00				
Suma:	0.21			0.19		0.03

Procedura ShannonFano(S)
1. $n \leftarrow |S|$
2. **Jeśli** $n = 2$ **to wykonaj**
3. dopisz 0 do kodu jednego elementu, a 1 do kodu drugiego elementu
4. **w przeciwnym razie jeśli** $n > 2$ **to wykonaj**
5. podziel S na dwa podzbiory S_1 i S_2 tak żeby $|P(S_1) - P(S_2)| \rightarrow \min$
6. rozszerz kod każdego symbolu w S_1 dopisując do niego 0
7. rozszerz kod każdego symbolu w S_2 dopisując do niego 1
8. **ShannonFano**(S_1)
9. **ShannonFano**(S_2)

Tab. 8.1: Procedura Shannona-Fano.

Ciąg S_{11} zawiera dwa elementy kod dla jednego z nich – E – rozszerzamy dodając kolejno zero, a do kodu D dodajemy jedynkę.

S_{11}	Prawdopodobieństwo	Kod	S_{11}	Prawdopodobieństwo	Kod	Różnica
E	0.09	000	D	0.12	001	
Suma:	0.09			0.12		0.03

Ciąg S_2 składa się również z dwóch elementów, a ich kody są odpowiednio przedłużane.

S_2	Prawdopodobieństwo	Kod	S_2	Prawdopodobieństwo	Kod	Różnica
B	0.21	10	A	0.39	11	
Suma:	0.21			0.39		0.18

Ostatecznie zatem mamy

S	Prawdopodobieństwo	Kod
A	0.39	11
B	0.21	10
C	0.19	01
D	0.12	001
E	0.09	000

Liczymy średnią długość kodu L

$$L_{SF} = 0.39 \cdot 2 + 0.21 \cdot 2 + 0.12 \cdot 3 + 0.09 \cdot 3 = 2.21$$

Uzyskany wynik wynosi tyle samo co w algorytmie Huffmana. Ogólnie tak jednak być nie musi. Im bliższe odwrotnością potęg dwójki są prawdopodobieństwa wystąpień symboli, tym efektywniejszy staje się algorytm Shannona-Fano, może on jednak co najwyżej dorównać algorytmowi Huffmana, ale nigdy go przewyższyć. Obydwa algorytmy opierają się na ukrytej w przesłanych wiadomościach nadmiarowości. Krótsze kody są przypisywane częściej spotykanym znakom, a dłuższe rzadziej występującym. Zmniejsza to rozmiar zakodowanej wiadomości i skraca czas jej transmisji. Wadami obu metod są: duża wrażliwość powstałych kodów na zakłócenia w transmisji oraz konieczność znania częstości występowania symboli przed rozpoczęciem kodowania.

9 Kodowanie Ziva-Lempla

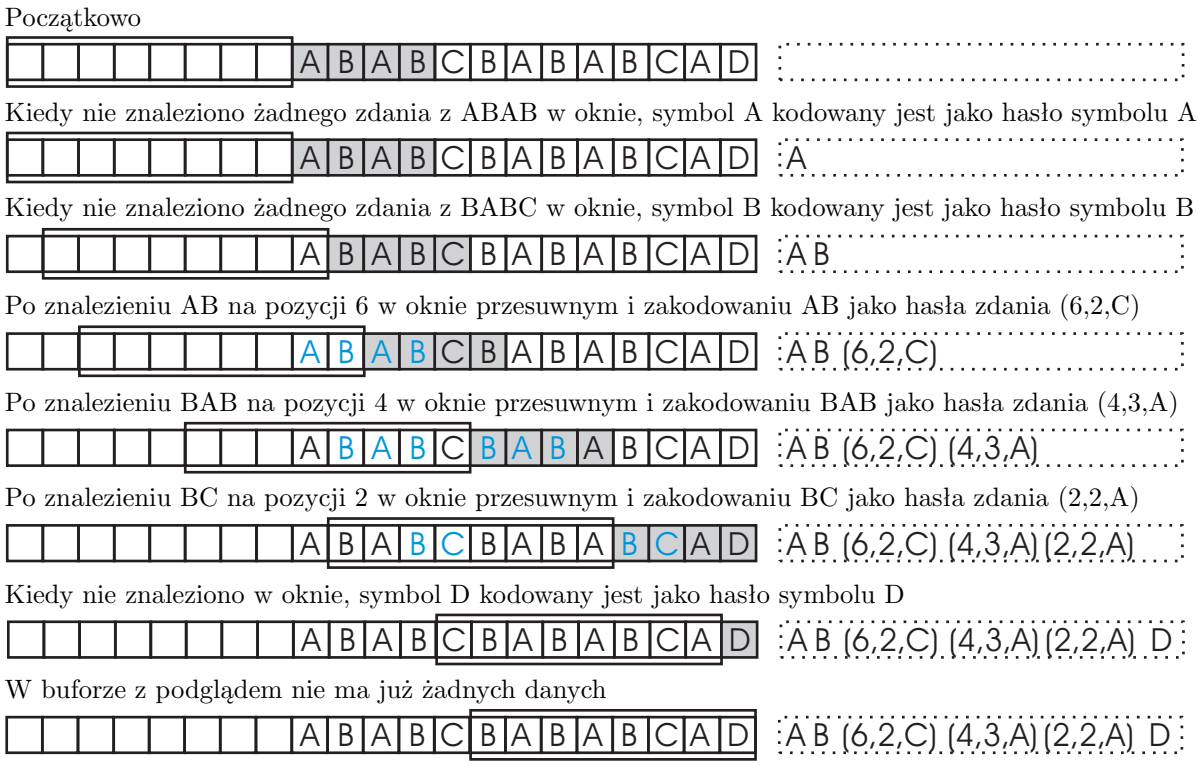
Kodowanie Ziva-Lempla opiera się na zasadzie przyporządkowania kodów ciągom symboli o zmiennej długości. W odróżnieniu od poprzednich metod nie wymagają informacji o rozkładzie prawdopodobieństwa występowania symboli w sygnale wejściowym. Oznacza to, że kodowanie obejmuje całą grupę pokrewnych metod bezstratnych. Są to metody słownikowe, których charakterystyczną cechą jest sukcesywne tworzenie w trakcie kodowania słowników zawierających ciągi symboli już przesłanych. Kolejno kodowane ciągi symboli zastępuje się informacją pozwalającą na identyfikację ciągu w słowniku.

Algorytmy z tej grupy są szeroko stosowane w programach pakujących pliki. W odróżnieniu od kodowania Huffmana, kodowanie Ziva-Lempla jest stosunkowo często wykorzystywana jako samodzielna technika kompresji obrazów.

W metodzie Ziva-Lempla wykorzystuje się *okno przesuwne* oraz *bufor z podglądem*. Zaczynając kompresję musimy sobie uświadomić, że mogą zajść dwie sytuacje między buforem a oknem: mogą istnieć pasujące do siebie zdania (zdanie – ciąg symboli występujących jeden po drugim np. *abc*, mamy zatem trzy symbole a, b, oraz c. Ciąg takich symboli będziemy nazywali zdaniem) o jakiejś długości lub zdania mogą do siebie nie pasować. Jeśli istnieje przynajmniej jedno dopasowanie, kodujemy najdłuższe z nich jako *hasło zdania*. Hasła zawierają trzy informacje: odstęp od brzegu okna przesuwnego, przy którym zaczyna się dopasowanie, liczbę pasujących symboli oraz pierwszy symbol z bufora zza dopasowania. Jeśli nie wystąpi żadne dopasowanie, kodujemy niedopasowany symbol jako *hasło symbolu*. Hasło symbolu zawiera tylko sam niedopasowany symbol, więc nie zachodzi w tym przypadku kompresja.

Gdy zostanie wygenerowane hasło kodujące n symboli, przesuwamy n symboli poza jeden brzeg okna przesuwnego i zastępujemy go z drugiej strony taką samą liczbą symboli z bufora z podglądem. Następnie uzupełniamy bufor danymi czekającymi na kompresję.

Rozważmy przykład kodowania dla ciągu $ABABCBABABCAD$ pokazany na Rys. 9.1. Przyjęto w nim, że okno przesuwne ma rozmiar 8 bajtów, a bufor podglądu danych ma rozmiar 4 bajtów.



Rys. 9.1: Ilustracja działania algorytmy Ziva-Lempla – faza kodowania. Kolorem szarym oznaczone jest podgląd bufora. Kolorem niebieskim oznaczone są symbole zgodne w oknie przesuwnym i buforze. W ramce z obwódką przerywaną przedstawione są przesyłane kody.

Oczywiście, żeby była lepsza kompresja możemy pominąć okalające nawiasy i przecinki w zapisach po skompresowaniu, czyli zamiast pisać np. $(6, 2, C)$ możemy zapisać po prostu $62C$. Dla przykładu rozważnego powyżej zysk w kompresji napisu $ABABCBABABCAD$ (13 znaków) jest nie wielki i wynosi zaledwie jeden znak – $AB62C43A22AD$ (12 znaków). Jednakże gdyby rozważany fragment byłby dłuższy prawdopodobnie osiągnęlibyśmy lepszy rezultat.

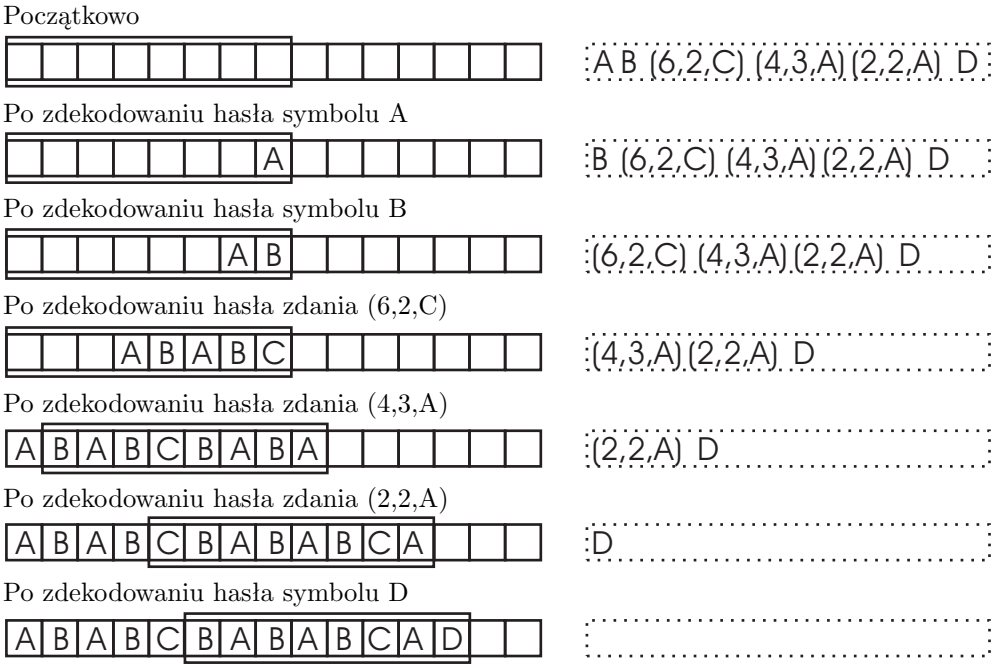
Dane dekompresuje się poprzez dekodowanie haseł i aktualizację okna przesuwnego, tak jak podczas kompresji. Gdy dekompresujemy poszczególne hasła, kopiujemy symbole kodowane przez to hasło do okna przesuwnego. Po odczytaniu hasła zdania, określamy odpowiednie przesunięcie w oknie i odszukujemy zdanie podanej długości. Gdy odczytujemy hasło symbolu, generujemy pojedynczy symbol zapisany w samym haśle. Procedura dekompresji została przedstawiona na Rys. 9.2.

Technika LZW została przyjęta jako metoda kompresji dla różnych formatów plików graficznych, np. GIF. Ponadto modyfikacja algorytmu LZW jest stosowana w implementacji modemu V.42 bis. Kompresja LZW w przeciwieństwie do Huffmana nie wymaga wyliczenia tablicy kodów przed rozpoczęciem pakowania. Jest ona adaptacyjna co oznacza że buduje słownik w locie podczas czytania danych wejściowych. Program dekodujący na podstawie sekwencji kodów w spakowanym pliku może ustalić, których pozycji słownika używał program kodujący, więc słownik nie musi być jawnie przekazywany.

Kodowanie Ziva-Lempla posiada wiele różnych odmian. Do najbardziej znanych należą:

LZ77 w tej technice dekoder otrzymuje dla każdego kodowanego ciągu symboli parę wartości: wskaźnik k oraz długość ciągu 1. Wskaźnik k adresuje ciąg jako podciąg w ciągu dotychczas zakodowanych symboli. Liczba k wskazuje, o ile symboli należy cofnąć się w ciągu poprzednio już odebranych symboli, aby znaleźć w nim podciąg o długości 1 identyczny z aktualnie kodowanym ciągiem. Koder wyszukuje w ciągu zakodowanych symboli ciąg do przesłania o największej długości 1. Jeżli $1 = 0$ wówczas zamiast k przesiane zostaną pojedyncze symbole. Wartości k mogą być kodowane zarówno ze stałą jak i ze zmienną liczbą bitów. W algorytmie zakłada się ograniczenia $k < K$ oraz $1 < L$ przy czym $L \ll K$ (typowo L = 16, K = 8192) Zastosowanie: w programach Squeeze, PKZIP, ZOO.

LZW powstał na bazie algorytmu LZ78 po wprowadzeniu poprawek przez Welcha. Przyjmuje się, że maksymalna długość ciągu L może być nieskończona i wprowadza się słownik ciągów poprzednio



Rys. 9.2: Ilustracja działania algorytmy Ziva-Lempla – faza dekodowania.

zakodowanych. Elementy tego słownika są ponumerowane. Algorytm rozpoczyna pracę ze słownikiem złożonym z wszystkich możliwych pojedynczych znaków. W trakcie kodowania do słownika dodawane są kolejne napotkane ciągi znaków, którym przypisuje się kolejne numery elementów słownika. Następnie przesyłane są one do dekodera. Dekodowanie jest jednoznaczne, gdyż dekoder rozpoczyna pracę z takim samym słownikiem, jak koder i rozbudowuje ten słownik w ten sam sposób. Zastosowanie: w programach ARJ, ARC, PAK.

LZSS niewielka modyfikacja algorytmu LZ77 polegająca na wykorzystaniu dodatkowego 1-bitowego przełącznika w przesyłanych danych i wprowadzeniu drzewiastej struktury danych poprawiającej efektywność kodowania.

LZPG odmiana algorytmu LZ78, w której wskaźnik adresuje wierzchołki drzewa reprezentującego strukturę słownika.

10 Kodowanie długości serii

Metodę kodowania długości serii (RLE – ang. *Run Length Encoding*) stosuje do kodowania powtarzających się po sobie tych samych symboli. Ciąg powtarzających się symboli koduje się jako parę $(s; n)$, gdzie n (kodowane za pomocą kodów ASCII) oznacza ilość powtórzeń, a s oznacza powtarzający się symbol. Takie kodowanie jest efektywne dla serii przynajmniej dwuelementowych, ponieważ dla pojedynczych symboli kod uzyskany na podstawie kodowania długości serii jest dwa razy dłuższy niż przy innych metodach. Takie pary powinno się więc kodować dla serii przynajmniej dwuelementowych. Dla odróżnienia, czy przesyłana jest seria czy nie, stosuje się tzw. znacznik p, czyli powtarzające się symbole koduje się jako trójkę $(p; s; n)$, przy czym p musi być tak dobrany, aby nie było symbolem s. Taką trójkę powinno się stosować do serii przynajmniej czteroelementowych.

Przykład 1 (kodowany znacznik n jest kodem ASCII): Załóżmy, że mamy następujący tekst składający się z 42 liter b, czyli *bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb*. Po zakodowaniu tego ciągu otrzymamy: $b*$. Gwiazdkę $(*)$ w tym przypadku odczytujemy jako wartość kodu ASCII i wynosi ona 42.

Przykład 2: Załóżmy, że mamy następujący tekst

222222222222222222227777777777777777999999999999999999999999

Po zakodowaniu tego ciągu otrzymamy:

221716926

W tym przypadku nie potrafimy rozróżnić, gdzie jest kodowany znak a gdzie zaczyna i kończy się liczba powtórzeń tego znaku. Można by zapis 22171926 zinterpretować np. tak: występuje dwukrotnie symbol 2, siedmiokrotnie symbol 1, dziewięciokrotnie symbol 1 i sześciokrotnie symbol 2 (oczywiście jest to jedna z możliwych interpretacji). Aby uniknąć takiej niejednoznaczności stosuje się przełącznik p. Jeśli przyjmiemy za znacznik p symbol x to nasz przykładowy tekst możemy zakodować w następujący sposób:

- n jako liczba

$$x221x719x926$$

- n jako kod ASCII, który reprezentuje liczbę powtórzeń danego symbolu

$$x2\S x7\blacktriangleright x9\rightarrow$$

lub bez znacznika x wtedy mamy (gdyż tutaj korzystając z kodowania ASCII mamy jednoznaczność, tzn. pierwszy znak to kodowany symbol drugi znak to kod ASCII reprezentujący liczbę powtórzeń danego symbolu):

$$2\S 7\blacktriangleright 9\rightarrow$$

gdzie:
§ – odpowiada kodowi ASCII o wartości 21;
▶ – odpowiada kodowi ASCII o wartości 19;
→ – odpowiada kodowi ASCII o wartości 26.

Uwaga: Wymienione kody ASCII o wartościach 21, 19 i 26 mogą mieć inną reprezentację graficzną w zależności od używanej czcionki w systemie.

Maksymalna długość serii, którą można reprezentować za pomocą trójki $(p; s; n)$ np. wynosi 255 dla 8 - bitowego kodu ASCII (w rzeczywistości 251, bo kodowanie rozpoczyna się dopiero od czterech powtórzeń). Kodowanie długości serii wykorzystywane jest między innymi w bazach danych, gdzie stałą długość rekordu uzyskuje się np. poprzez dopełnianie zerami oraz przy kodowaniu czarno - białych obrazów przesyłanych faksem.

Literatura

[1] L.Banachowski i in. *Algorytmy i struktury danych*; WNT, 1996.

[2] T.H.Cormen i in. *Wprowadzenie do algorytmów*, WNT, 2000

[3] A.Drozdek *Struktury danych w jezyku C*, WNT, 1996

[4] K.Loudon *Algorytmy w C*; HELION, 1999.

[5] D.E.Knuth *Sztuka programowania*, WNT, 2002

[6] R.Sedgewick *Algorytmy w C++*, Wydawnictwo RM, 1999

[7] P.Wróblewski, *Algorytmy, struktury danych i techniki programowania*, HELION, 1996

[8] N. Wirth, *Algorytmy + struktury danych = programy*, WNT, 2001