ROZDZIAŁ 3. METODA HUFFMANA

Spośród wielu metod bezstratnego kodowania algorytm Huffmana jest chyba najbardziej znany i popularny, głównie ze względu na takie cechy jak prostota, łatwość aplikacji i przede wszystkim efektywność tworzenia kodu. Jest to optymalny koder tworzący kod symboli. Znaczy to, że jeśli sekwencja kodowa tworzona jest przez przypisanie każdemu symbolowi alfabetu źródła określonego słowa kodowego i kolejnym emitowaniu słów kodowych odpowiadających pojawiającym się symbolom w strumieniu wejściowym, to wówczas metoda Huffmana daje najkrótszy możliwy kod.

Prace D.A. Huffmana [1] w dziedzinie kodowania są bardzo znaczące. Dał on początek metodzie kodowania, która została opisana w najczęściej cytowanej publikacji z teorii informacji [2] i jest powszechnie stosowana do dziś. Z dalszych prac nad tą metodą warto wymienić przede wszystkim szereg opracowań dotyczących adaptacyjnej wersji kodera Huffmana z lat siedemdziesiątych [3,4] i osiemdziesiątych [5,6], aż do współczesnych rozwiązań [7,8]. Świadczy to o rozwoju i dużej popularności tej metody przez dziesiątki lat.

W rozdziale tym przedstawiona została na początku metoda kodowania znana jako metoda Shannona-Fano oraz w części zasadniczej metoda Huffmana, także w przydatnej do praktycznych zastosowań wersji adaptacyjnej. Zwrócono także uwagę na kilka istotnych problemów występujących przy implementacji tych algorytmów. Korzystano przy tym z niektórych cennych uwag przedstawionych w [8].

3.1. Kody o minimalnej długości

Metoda Shannona-Fano

Jest to właściwie pierwsza dobrze znana metoda kodowania, opracowana na podstawie teorii informacji w celu minimalizacji długości kodu przypisanemu wejściowemu strumieniowi symboli. Chodzi o stworzenie kodu o minimalnej nadmiarowości w stosunku do ilości informacji obliczonej przy statystycznym modelu informacji. Mając prawdopodobieństwa wystąpienia każdego z symboli alfabetu w kodowanym strumieniu danych, metoda opracowana niezależnie przez Shannona i Fano zmierza w kierunku konstrukcji kodu o kilku interesujących ogólnych własnościach:

- kody przyporządkowywane są pojedynczym symbolom i mają różną liczbę bitów,
- kody symboli o większym prawdopodobieństwie wystąpienia w strumieniu są krótsze (mają mniej bitów) od kodów symboli o mniejszym prawdopodobieństwie,
- tworzony kod jest oczywiście jednoznacznie dekodowalny.

Pierwsze dwie cechy są ze sobą związane i wypływają bezpośrednio z intuicyjnego rozumienia informacji w ujęciu statystycznym oraz jej miary. Trzeci warunek jest koniecznym w każdej metodzie bezstratnego kodowania.

W metodzie tej zastosowano strukturę drzewa binarnego. Pozwala to z jednej strony prosto zrealizować jednoznaczną dekodowalność, z drugiej zaś poprzez umieszczenie bliżej korzenia symboli występujących częściej (o większym prawdopodobieństwie) w strumieniu danych, a symboli występujących rzadziej - dalej od korzenia drzewa (na niższym poziomie) umożliwia realizację pierwszych dwóch własności kodu. Dla wygody opisu przyjęto odwrotną konwencję rysowania drzewa, tj. korzeń na górze - liście na dole.

Drzewo takie składa się z korzenia, węzłów (wewnętrznych oraz zewnętrznych - liści) znajdujących się na kolejnych poziomach drzewa oraz gałęzi łączących poszczególne węzły ze sobą, a także węzły najwyższego poziomu z korzeniem. Z liśćmi drzewa związane są symbole alfabetu opisującego źródło danych wejściowych. Wszystkie węzły pomiędzy korzeniem a liśćmi to węzły wewnętrzne.

Pozostaje więc jedynie określić sposób budowania takiego drzewa oraz związanego z nim sposobu wyznaczania słów kodowych. Shannon i Fano zaproponowali następującą metodę:

Algorytm 3.1A. Kodowanie metodą Shannona-Fano

1. Określenie prawdopodobieństwa wystąpienia w kodowanym strumieniu danych wszystkich symboli alfabetu (na podstawie częstości wystąpień tych symboli w strumieniu); czasami zwłaszcza dla małolicznych zbiorów wygodniej jest mówić o wagach symboli równych liczbie ich wystąpień w strumieniu.
2. Sortowanie listy symboli, w zależności od prawdopodobieństw ich wystąpienia (częstości), w kolejności od najbardziej prawdopodobnych (na górze) do najmniej prawdopodobnych (na dole listy).
3. Podział listy symboli na dwie grupy (z zachowaniem pozycji symbolu na liście, tj. górna grupa zawiera częściej występujące symbole według klasyfikacji z p.2, podczas gdy dolna grupa obejmuje rzadziej pojawiające się symbole) o możliwie bliskiej sumie wystąpień symboli z górnej części i sumie wystąpień symboli z dolnej części.
4. Przyporządkowanie górnej grupie symboli binarnej cyfry '0', a dolnej - '1'.
5. Rekursywne powtarzanie kroków 3 i 4 dla utworzonych przez ostatni podział obydwu list symboli (części górnej i dolnej), dodawanie kolejnych cyfr binarnych do kodów poszczególnych symboli (są to bity przypisywane sukcesywnie grupom, do których został zakwalifikowany symbol) aż do momentu, gdy wszystkie grupy utworzone na skutek kolejnych podziałów są jednoelementowe. Oznacza to, że dotarliśmy do wszystkich liści tworzonego drzewa i każdy z symboli alfabetu ma całkowicie określone słowo kodowe. Tworzą go kolejne bity przypisane grupom, do których trafił symbol przy pierwszym podziale (najstarszy bit słowa), drugim, itd., aż to określenia symbolu jako grupy jednoelementowej (bit tej grupy stanowi najmniej znaczący bit słowa).
6. Przypisanie kolejnym symbolom strumienia wejściowego odpowiednich słów kodowych, składających się w bitową sekwencję wyjściową kodera.

Algorytm 3.1B. Dekodowanie metodą Shannona-Fano (S-F)

1. Pobranie ze zbioru danych zakodowanych informacji na temat prawdopodobieństwa występowania w zbiorze poszczególnych symboli alfabetu.
2. Zbudowanie binarnego drzewa kodowego analogicznie jak w procesie kodowania.
3. Pobieranie kolejnych bitów, organizowanie ich w słowa kodowe i przeszukiwanie drzewa w celu znalezienia kolejnych liści - dekodowanych symboli alfabetu.

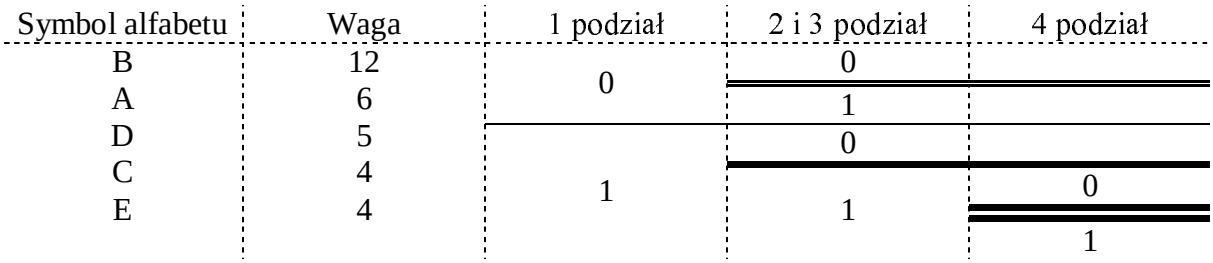
PRZYKŁAD 3.1. Kodowanie metodą S-F.

Rozważmy prosty przykład tworzenia binarnego kodu metodą Shannona-Fano dla zbioru danych o pięcioliterowym alfabecie: A, B, C, D, E. Częstość występowania poszczególnych symboli w tym zbiorze przedstawia tabela 3.1.

Tabela 3.1. Symbole alfabetu przykładowego zbioru danych wraz z ich częstością wystąpień.

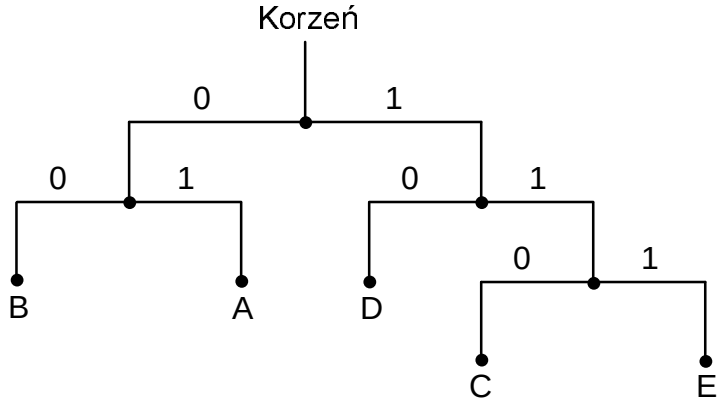
Symbol	A	B	C	D	E
Częstość wystąpień	6	12	4	5	4

Lista symboli sklasyfikowanych w kolejności malejącego prawdopodobieństwa oraz kolejne podziały tej listy według algorytmu 3.1A przedstawione zostały na rys. 3.1.



Rys. 3.1. Przykład realizacji algorytmu Shannona-Fano. Symbole rozdzielone są liniami o różnej grubości, które obrazują kolejne podziały listy symboli: —— - pierwszy podział, ═══ - drugi podział, ▬▬ - trzeci podział, ▬▬ - czwarty podział.

Słowa kodowe przypisane poszczególnym symbolom wyglądają następująco:
A - 01, B - 00, C - 110, D - 10, E - 111. Efektywność tak skonstruowanego kodu została określona w tabeli 3.2. Opowiadający tym słowom kodowym schemat drzewa binarnego przedstawia rys.3.2.



Rys. 3.2. Binarne drzewo kodowe dla prostego zbioru danych z przykładu 3.1 opisanego alfabetem pięciu symboli A, B, C, D, E, utworzone zgodnie z metodą S-F.

Tabela 3.2. Zestawienie ilości informacji związanych z wystąpieniem poszczególnych symboli alfabetu zbioru z przykładu 3.1 z uzyskaną efektywnością kodu S -F.

Symbol	Oszacowane prawdopodo-bieństwo	Ilość informacji własnej symboli [bity/symbol]	Suma informacji związana z występo-waniem symbolu [bity]	Długość słowa kodowego S-F [bity]	Suma bitów reprezentacji S-F
A	6/31	2,369	14,215	2	12
B	12/31	1,369	16,431	2	24
C	4/31	2,954	11,817	3	12
D	5/31	2,632	13,161	2	10
E	4/31	2,954	11,817	3	12

Suma	67,44		70

Z zestawienia przedstawionego w tabeli 3.2 wynika, iż długość nowej bitowej reprezentacji przykładowego zbioru danych (70 bitów) jest większa od ilości informacji zawartej w zbiorze (przyjęliśmy prosty model źródła bez pamięci) wyrażonej w bitach (67,44). Uzyskany kod jest więc nadmiarowy. Przyczyną tej nadmiarowości jest fakt, iż algorytm S-F nie w pełni realizuje zadania postawione przed algorytmem, który ma osiągać minimalną długość kodu.

Przykładowo dla symbolu A ilość informacji związana z jego wystąpieniem wynosi 2,32 bita. Należałoby więc przyporządkować symbolowi A kod o długości powiedzmy 2,3 bita. Nie jest to niestety możliwe w metodzie S-F. Aby uzyskać kod dokładnie o tych własnościach należałoby opracować technikę przydzielającą symbolom kody o niecałkowitej ilości bitów. Dopiero bowiem wtedy można by dokładnie odwzorować ilość informacji związanej z wystąpieniem poszczególnych symboli w słowa kodowe o odpowiedniej długości. Zrealizować tego postulatu nie pozwala także nieco efektywniejszy algorytm Huffmana.

Metoda Huffmana

Koncepcja kodowania metodą Huffmana opiera się na takich samych założeniach jak metoda Shannona-Fano i jest próbą skuteczniejszej realizacji metody kodowania o wspomnianych trzech własnościach. Podobnie jak w metodzie S-F, tworzone są bitowe słowa kodowe o zmiennej długości przyporządkowane poszczególnym symbolom alfabetu. Podstawową różnicą jest odwrotny sposób budowania drzewa w algorytmie kompresji, a mianowicie od liści w kierunku korzenia. Łączone są ze sobą kolejne węzły zaczynając od najniższych poziomów drzewa, powstają nowe węzły na wyższych poziomach, które stopniowo zbiegają ostatecznie do korzenia.

Algorytm kodowania metodą Huffmana przedstawia się następująco:

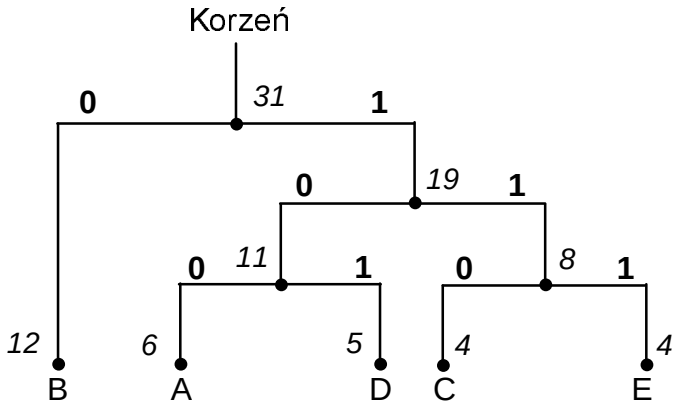
Algorytm 3.2. Kodowanie metodą Huffmana

1. Określenie wag wszystkich symboli alfabetu na podstawie częstości wystąpień tych symboli w strumieniu. Symbole te stanowią początkową warstwę węzłów budowanego drzewa binarnego, tzw. węzłów wolnych, które mogą być łączone w węzły rodzicielskie wyższego poziomu.
2. Sortowanie listy symboli według wag: najbardziej prawdopodobne na górze, najmniej prawdopodobne na dole listy.
3. Odszukanie dwóch wolnych węzłów z najniższymi wagami i utworzenie z nich nowego węzła-rodzica, z wagą równą sumie wag węzłów łączonych-dzieci.
4. Dodanie nowego węzła (rodzica) do listy węzłów wolnych, oraz usunięcie z tej listy dwóch węzłów połączonych (dzieci).
5. Przypisanie gałęziom prowadzącym od rodzica do węzłów dziecięcych odpowiednich cyfr binarnych - '0' i '1' (np. lewa gałąź - ‘0’, prawa - ‘1’).
6. Powtarzanie kroków 3-5 aż do momentu, gdy pozostanie tylko jeden wolny węzeł - jest to korzeń drzewa.
7. Odczytanie ze struktury drzewa słów kodowych dla kolejnych liści drzewa (symboli alfabetu źródła danych). Słowo stanowią bity odczytane dla kolejnych gałęzi drzewa przy przejściu od korzenia do liścia. Cyfra binarna przypisana gałęzi odchodzącej bezpośrednio

od korzenia jest najbardziej znaczącym bitem słowa, podczas gdy bit skojarzony z gałęzią dochodzącą bezpośrednio do liścia - najmniej znaczący.

8. Przypisanie kolejnym symbolom strumienia wejściowego odpowiednich słów kodowych, składających się w bitową sekwencję wyjściową kodera.

Proces dekodowania przebiega dokładnie odwrotnie, analogicznie jak w metodzie S-F (algorytm 3.1.B). Prześledźmy działanie algorytmu Huffmana na przykładzie zbioru danych kodowanego metodą S-F - przykład 3.1. Sposób budowania binarnego drzewa kodowego przy pomocy algorytmu 3.2 został przedstawiony na rys. 3.3.



Rys. 3.3. Binarne drzewo kodowe Huffmana dla przykładu 3.1. Pochyloną czcionką oznaczone są wagi poszczególnych węzłów, a pogrubioną - kolejne bity przydzielane poszczególnym gałęziom drzewa.

Sekwencje kodowe przyporządkowane poszczególnym symbolom alfabetu są odmienne niż w przypadku metody S-F i przedstawiają się następująco: A - 100, B - 0, C - 110, D - 101, E - 111. Tak więc najczęściej występującemu symbolowi B przyporządkowane zostało jednobitowe słowo kodowe, podczas gdy wszystkim innym - sekwencje trójbitowe. Intuicyjnie wydaje się to słuszniejsze od sekwencji uzyskanych metodą S-F, gdyż waga symbolu B jest zdecydowanie różna od wag pozostałych symboli, które z kolei są dużo bardziej zbliżone. Porównując wyniki z Tabeli 3.2 i Tabeli 3.3, mówiącej o efektywności uzyskanego kodu Huffmana można stwierdzić, iż rzeczywiście uzyskano mniejszą długość nowej reprezentacji w przypadku metody Huffmana. Jakkolwiek zysk wydaje się niezbyt wielki (1 bit), to jednak przy pomocy kodu Huffmana udało się bardziej zbliżyć do granicznej wartości entropii.

Tabela 3.3. Efektywność kompresji metodą Huffmana.

Symbol	Oszacowane prawdopodo-bieństwo	Ilość bitów słowa kodowegodu Huffmana	Suma bitów reprezentacji Huffmana
A	6/31	3	18
B	12/31	1	12
C	4/31	3	12
D	5/31	3	15
E	4/31	3	12

Suma	69

Warto zauważyć, że sposób konstrukcji słów kodowych w obu metodach nie zależy od kolejności wystąpienia tych symboli w strumieniu wejściowym. A przecież jeśli symbole będą się pojawiać w kolejności: 6×A, 12×B, 4×C, 5×D i 4×E, to wystarczy zakodować według schematu RLE liczbę wystąpień na czterech bitach i symbol na trzech bitach, aby uzyskać 35 bitową sekwencję kodową jednoznacznie dekodowalną, znacznie krótszą od 69-bitowej reprezentacji Huffmana. Widać tutaj poważne ograniczenie metod budowanych na bazie źródeł bez pamięci.

Złożoność algorytmów: Huffmana i Shannona-Fano jest zbliżona (wymagają do realizacji zbliżonych mocy obliczeniowych), ale efektywność kodowania (w znaczeniu minimalnej długości nowej bitowej reprezentacji oryginalnego zbioru danych) metody Huffmana jest zawsze co najmniej równa efektywności metody S -F.

Kodowanie metodą Huffmana daje najkrótszy kod ze wszystkich metod entropijnych (na bazie entropii - statystycznej miary informacji) przypisujących poszczególnym symbolom alfabetu źródła sekwencje kodowe o całkowitej liczbie bitów. Duża efektywność tej metody przy jednoczesnej dużej prostocie stanowi o jej popularności i szerokim obszarze zastosowań. Jednak w obu przedstawionych algorytmach kodowania (Shannona-Fano i Huffmana) konieczne jest dopisanie do wyjściowej sekwencji bitowej informacji o modelu źródła informacji, który wykorzystano, tj. mapy częstości wystąpień każdego z symboli alfabetu. Dopiero wtedy dekoder potrafi zbudować dokładnie takie samo binarne drzewo kodowe jak w koderze, jednoznacznie określające słowa kodowe poszczególnych symboli. Warunkiem koniecznym jest oczywiście ten sam sposób realizacji algorytmu 3.2, tj. niedookreślonego sposobu sortowania, tzn. kolejności ustawienia symboli o tej samej wadze, decyzji wyboru dwóch wolnych węzłów z najniższymi wagami w sytuacji, gdy węzłów z najniższą wagą jest więcej, a nawet konwencji przypisania zer i jedynek poszczególnym gałęziom drzewa.

Związane z koniecznością dopisania mapy częstości wydłużenie długości wyjściowej reprezentacji powoduje spadek efektywności kodowania szczególnie dla małych zbiorów danych o dużym alfabecie. Takie rozwiązanie nazywane jest statycznym (lub według p. 2.2. - póładaptacyjnym), gdyż taki sam model statystyczny wykorzystywany jest dla całego strumienia kodowanych symboli. Można wyobrazić sobie rozwiązanie adaptacyjne, kiedy do w zależności od cech lokalnych zbioru używane są różne modele. Jednak konieczność dopisania każdego z użytych modeli do pliku wyjściowego (dla adaptacji wstecz) ogranicza znacznie jego skuteczność. Poniżej przedstawiono rozwiązanie, które umożliwia realizację kodera Huffmana w wersji adaptacyjnej w oparciu o modyfikowaną strukturę drzewa binarnego, bez konieczności dopisywania jakiejkolwiek informacji dodatkowej.

3.2. Adaptacyjny koder Huffmana

Okazuje się, że w pewnych przypadkach statyczny algorytm Huffmana jest bezradny. Jeśli chcemy zakodować strumień danych generowany przez źródło o jednakowo prawdopodobnych symbolach alfabetu, to przydzielone im w drzewie Huffmana słowa kodowe będą tej samej długości. W przypadku, gdy oryginalna reprezentacja danych nie była nadmiarowa np. po dwa bity na każdy symbol czteroelementowego alfabetu, wówczas utworzenie kodu Huffmana nie daje żadnej korzyści zmniejszenia długości zbioru oryginalnego.

Aby uzyskać w takim przypadku efekt kompresji należy wykorzystać fakt, iż lokalnie statystyka zbioru kodowanego musi być nierównomierna. Trzeba więc zastosować adaptacyjny algorytm dopasowania modelu statystycznego do lokalnej charakterystyki danych. Aby uzyskać modyfikację drzewa Huffmana dla kodowanego strumienia danych w zależności od lokalnej statystyki można sobie wyobrazić generalnie taki scenariusz. Pewna początkowa statystyka wystąpień poszczególnych symboli alfabetu jest modyfikowana po zakodowaniu kolejnego symbolu poprzez zwiększenie o jeden liczby jego wystąpień. Może się jednak zdarzyć, że większa waga węzła powoduje utratę przez drzewo binarne charakteru drzewa Huffmana. Trzeba by więc przeprowadzić konstrukcję nowego drzewa dla symboli o zmienionych wagach. Taki sam proces musi być powtarzany przez dekoder w celu uzyskania jednoznacznie dekodowalnego kodu. Adaptacyjność jest więc okupiona większymi kosztami obliczeniowymi, związanymi z konstruowaniem nowego drzewa po każdym zakodowanym symbolu.

Lepszą koncepcją okazuje się odpowiednia modyfikacja jedynie części drzewa tak, aby zachowało ono swój charakter drzewa Huffmana. Prosty zbiór reguł konstrukcji i adaptacyjnej modyfikacji drzewa Huffmana może wyglądać następująco:

Algorytm 3.3. Adaptacyjna modyfikacja drzewa Huffmana.

1. Punktem wyjścia jest aktualna postać drzewa Huffmana odzwierciedlająca statystykę kodowanych dotąd danych ze strumienia wejściowego. Węzły utworzonego drzewa Huffmana ułożone są na kolejnych poziomach według ich odległości od korzenia.
2. Wyznaczenie listy zawierającej uporządkowane węzły drzewa w kolejności od najniższego poziomu do najwyższego, a na każdym poziomie w porządku rosnących wag (jest to możliwe w każdym przypadku dla drzewa Huffmana). Poziom pierwszy to najbardziej odległe od korzenia węzły zewnętrzne najrzadziej występujących symboli o najmniejszych wagach, którym przypisane są najdłuższe słowa kodowe. Kolejne poziomy to węzły o rosnących wagach i coraz krótszych słowach kodowych aż do poziomu najwyższego, na którym znajduje się korzeń drzewa. Węzły na każdym poziomie, porządkowane według rosnących wag, ustawiane są w konwencji od lewej do prawej.
3. Modyfikacja drzewa Huffmana po wystąpieniu każdego pojedynczego symbolu w strumieniu wejściowym odbywa się według następujących zasad:
   a. inkrementacja wagi węzła symbolu: $w'=w+1$,
   b. zamiana węzłów: jeśli na liście węzłów za węzłem o zwiększonej wadze w' występują węzły z wagą w, to dokonujemy zamiany tego węzła z ostatnim węzłem o wadze w,
   c. kontynuacja modyfikacji w górę drzewa: inkrementacja wagi rodzica i ewentualne nowe zamiany według zasady przedstawionej w poprzednim podpunkcie b), aż do poziomu korzenia.

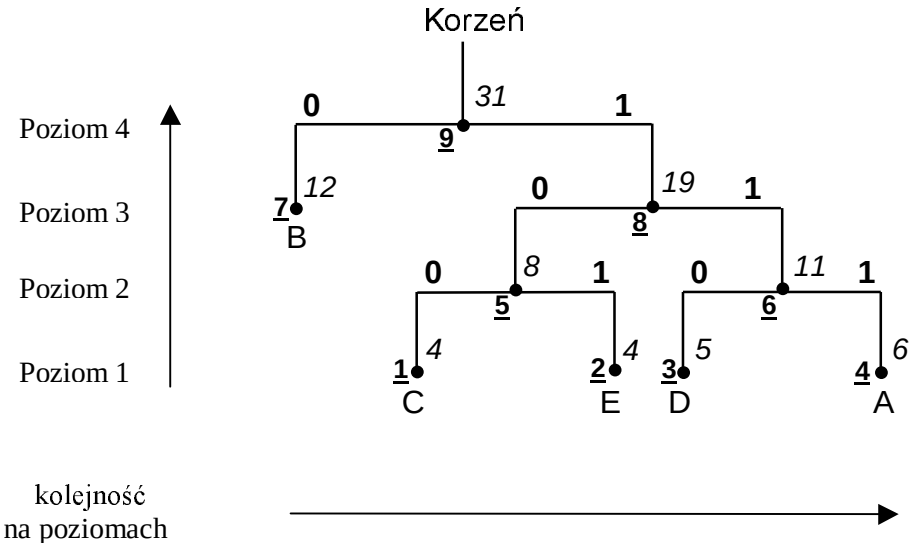
Rozważmy binarne drzewo kodowe Huffmana danych z przykładu 3.1. Poziomy drzewa i kolejność ustawienia węzłów na liście przedstawia rys. 3.4.

Aby prześledzić proces modyfikacji drzewa Huffmana rozważmy przykład 3.2.

PRZYKŁAD 3.2. Kodowanie metodą Huffmana w wersji adaptacyjnej.

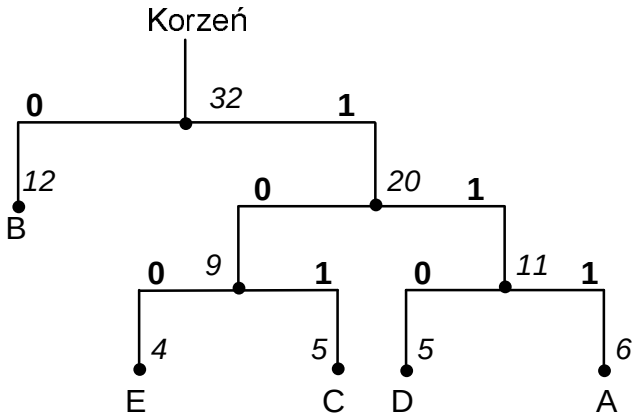
W kodowanym strumieniu danych o pięcioznakowym alfabecie liczba wystąpień każdego z symboli jest następująca: $w_A = 6$, $w_B = 12$, $w_C = 4$, $w_D = 5$, $w_E = 4$ (jak w przykładzie 3.1). Drzewo Huffmana skonstruowane na podstawie tych wystąpień wygląda jak na rys. 3.4. Na wejściu kodera pojawia się teraz ciąg kilku symboli C. Zaobserwujmy zmiany zachodzące

w drzewie Huffmana adaptującym się do bieżącej statystyki danych. Są one przedstawione na kolejnych rysunkach 3.5 - 3.8.



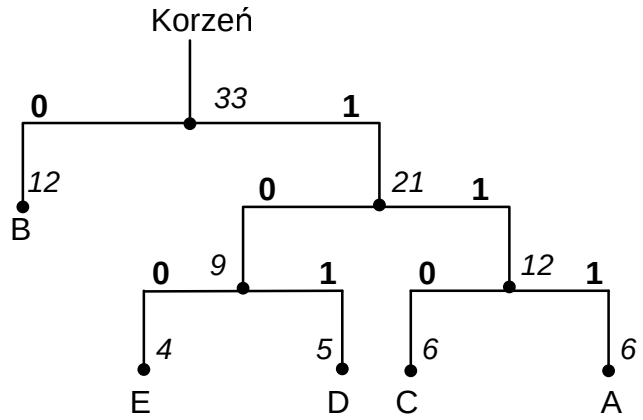
Rys. 3.4. Drzewo Huffmana dla danych z przykładu 3.1 w konwencji ustalonej przez adaptacyjny algorytm modyfikacji drzewa w trakcie kodowania strumienia danych (algorytm 3.3). Pochyloną czcionką oznaczone są wagi poszczególnych węzłów, pogrubioną bity przydzielone gałęziom, a podkreśloną – pozycję węzła na liście.

Gdy w strumieniu wejściowym pojawia się pierwszy z ciągu symbol C, jest on kodowany zgodnie z drzewem z rys. 3.4 przy pomocy słowa kodowego 100, a następnie następuje aktualizacja drzewa kodowego. Waga symbolu C rośnie o 1 (punkt 3a algorytmu 3.3), po czym następuje zamiana miejscami symboli C i E zgodnie z podpunktem 3b algorytmu 3.3. Zmodyfikowane drzewo wygląda jak na rys. 3.5, przy czym zgodnie z podpunktem 3c tego algorytmu wagi odpowiednich węzłów rodzicielskich wzrosły o jeden, aż do korzenia. Oczywiście, przy każdej zamianie węzłów modyfikowane są słowa kodowe przypisane liściom.



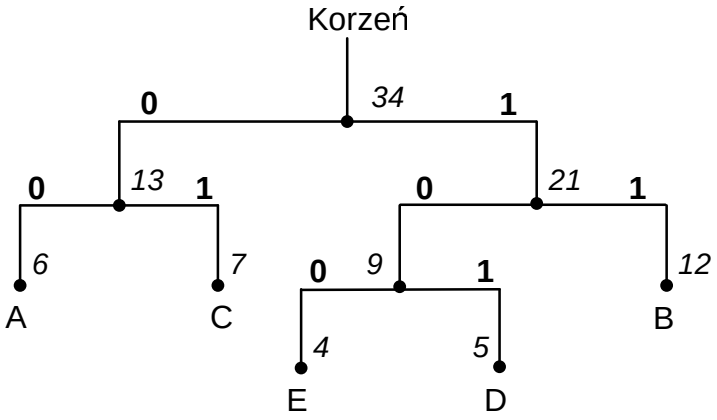
Rys. 3.5. Modyfikacja drzewa z rys. 3.4 po pojawieniu się jednego symbolu C w strumieniu wejściowym.

Po następnym symbolu C na wejściu, kodowanym sekwencją bitową 101, i analogicznej zamianie symboli C i D kształt drzewa nie ulega zmianie (rys.3.6). Zmieniają się jedynie słowa kodowe tych symboli oraz wagi kolejnych rodziców, aż do korzenia.



Rys. 3.6. Modyfikacja drzewa z rys. 3.5 po kolejnym pojawieniu się symbolu C w strumieniu wejściowym.

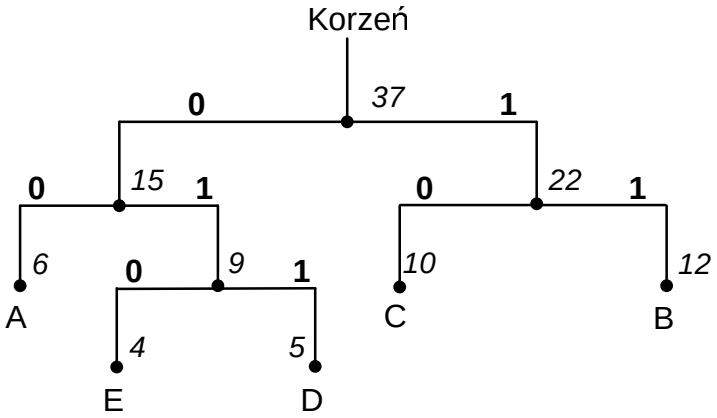
Kolejna wartość C w strumieniu wejściowym powoduje duże zmiany w kształcie drzewa. Zwiększenie wagi symbolu C do wartości 7 nie zapowiada jeszcze żadnej rewolucji w kształcie drzewa. Według przyjętego porządku węzłów kolejny węzeł - liść symbolu A - ma co prawda niższą wagę, ale już następny na liście węzeł - wewnętrzny z wagą 9 - ma wagę wyższą. Wobec tego zamieniamy miejscami symbole C i A nie zmieniając kształtu drzewa. Modyfikując wagę rodzica obu tych węzłów na wartość 13 można zauważyć, że ma on wyższą wagę niż występujący po nim węzeł wyższego poziomu - liść symbolu B. Zamieniamy więc ze sobą te węzły, przy czym za węzłem rodzicielskim idą także wszystkie odgałęzienia drzewa w dół. Drzewo przyjmuje nowy kształt z rys. 3.7. Jednocześnie słowo kodowe dla symbolu C (a także symbolu A) zostaje skrócone z trzech do dwóch bitów kosztem najczęściej występującej dotąd literki B, której słowo jest wydłużone z jednego do dwóch bitów.



Rys. 3.7. Modyfikacja kształtu drzewa z rys. 3.6 po kolejnym C na wejściu.

Następna zmiana kształtu drzewa wystąpi po trzykrotnym wystąpieniu literki C. Wówczas liść symbolu C z wagą 10 zostanie zamieniony miejscami z następnym w kolejności węzłem wewnętrznym - rodzicem węzłów z symbolami E i D - rys. 3.8. W tym przypadku zmiana

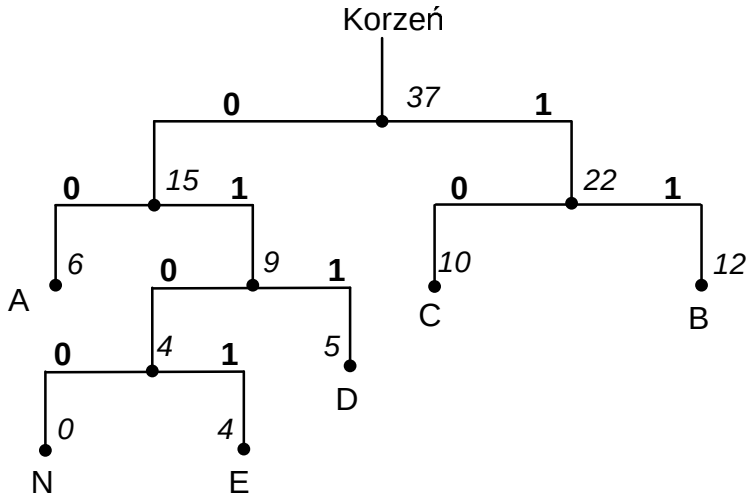
kształtu drzewa nie wpływa na długość słowa kodowego symbolu C, ponieważ jego węzeł pozostaje na tym samym poziomie drzewa.



Rys. 3.8. Modyfikacja drzewa z rys. 3.7 po odczytaniu trzech następnych symboli C w strumieniu wejściowym.

Dalsza modyfikacja drzewa przebiega analogicznie. Mechanizm dodania nowego węzła zewnętrznego w przypadku pojawienia się nowego symbolu przedstawia rys. 3.9. Węzeł o najmniejszej wadze staje się węzłem wewnętrznym, do którego dołączone są, jako węzły podrzędne, nowy liść z symbolem, który się właśnie pojawił oraz stary liść z symbolem, który do tego momentu miał najmniejszą wagę. Zostają przy tym zachowane wszystkie własności drzewa Huffmana przyjęte w algorytmie adaptacyjnym.

Wracając do rozważanego przykładu załóżmy, że na wejściu pojawił się symbol nie reprezentowany dotąd w drzewie, np. N. Modyfikacja drzewa Huffmana przedstawia się następująco:



Rys. 3.9. Modyfikacja drzewa z rys. 3.8 w wyniku pojawieniu się na wejściu nowego symbolu N. W kolejnym kroku waga symbolu N jest inkrementowana.

Praktyczna realizacja kodera Huffmana wymaga rozwiązania wielu dodatkowych problemów. Na niektóre z nich zwrócono uwagę w następnym podrozdziale.

3.3. Wybrane problemy aplikacyjne adaptacyjnego kodera Huffmana

Aby lepiej zrozumieć problemy występujące przy realizacji kodera Huffmana zostanie poruszonych kilka istotnych w implementacji zagadnień. Obok użytecznych struktur danych i procedur kodowania trzeba zwrócić uwagę na ograniczenia rozmiaru stosowanych struktur danych, przepełnienie drzewa, konieczność przeskalowania drzewa czy też oszczędną formę adaptacji drzewa Huffmana. Można uzyskać także bardziej przydatny do zastosowań transmisyjnych kod Huffmana, który powstaje poprzez dookreślenie algorytmu 3.2.

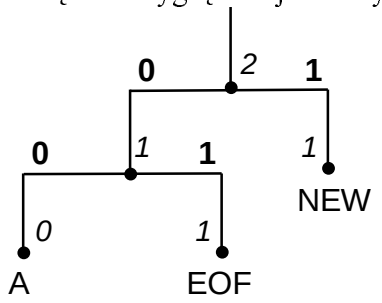
Istotną sprawą jest także początkowa postać adaptacyjnego drzewa Huffmana, czyli inicjalizacja adaptacyjnego algorytmu Huffmana. Możliwych jest wiele rozwiązań. Znając specyfikę źródła informacji początkowy model liczby wystąpień poszczególnych symboli może być dobrany odpowiednio, jako informacja dostępna *a priori*, bez konieczności dopisywania mapy zliczeń do pliku wyjściowego (ewentualnie tylko symbol określający typ modelu ze zbioru dostępnych w danym koderze/dekoderze). Bez tej wiedzy dostępnej w koderze trzeba przepisać inicjacyjne wagi symboli alfabetu w procesie kodowania na wyjście zwiększając długość kodowej reprezentacji.

Innym rozwiązaniem jest bardzo uboga początkowa postać drzewa Huffmana, zawierająca jedynie dwa symbole: znak końca (EOF) oraz znak nowego symbolu (NEW). Takie rozwiązanie stosowane jest w koderach do zastosowań uniwersalnych - kompresji dowolnych zbiorów o nie znanej wcześnie charakterystyce. Znak końca umieszczany jest w strumieniu wyjściowym jako ostatni, po zakodowaniu wszystkich symboli z wejścia. Jest to znak dla dekodera, że proces odtwarzania reprezentacji oryginalnej został zakończony. Można oczywiście ten sam efekt zatrzymania dekodera uzyskać poprzez umieszczenie liczby kodowanych danych w nagłówku pliku wyjściowego. Rozwiązanie to jest jednak mniej eleganckie, nie wiadomo bowiem jak duży może być kodowany zbiór danych, czyli ile bajtów przeznaczyć na zapis jego długości. Inicjacja drzewa kodowego jedynie z dwoma symbolami o wadze 1 daje dużą oszczędność (jednobitowe słowa kodowe) poprzez nie przydzielanie słów kodowych symbolom, które nie wystąpiły dotąd w strumieniu wejściowym. Często się tak zdarza, że nie cała dynamika reprezentacji oryginalnej jest wykorzystana w zbiorze danych, więc umieszczenie takich symboli w drzewie kodowym rezerwuje słowa, które nigdy nie zostaną użyte i niepotrzebnie wydłuża słowa symboli kodowanych.

Gdy na wejściu pojawi się symbol, którego nie ma w drzewie, wtedy koder emituje słowo przypisane symbolowi NEW, a następnie nowy symbol w oryginalnej reprezentacji. Aktualizacja drzewa (w koderze i dekoderze) poprzedzona jest w tym przypadku procedurą umieszczenia nowego symbolu w drzewie. Polega ona na wpisaniu na początek listy węzłów dwu nowych: jako pierwszego liścia nowego symbolu o wadze 0 i jako trzeciego - rodzica tegoż nowego liścia oraz węzła znajdującego się dotąd na pierwszym miejscu listy węzłów z najniższą wagą. Tak więc kolejność węzłów na liście po tej procedurze wygląda następująco:

- liść nowego symbolu z wagą 0,
- liść symbolu z najmniejszą dotąd wagą $w_{\min}$,
- węzeł wewnętrzny, rodzic obu tych liści z wagą $w_{\min}$,
- kolejne węzły przesunięte o dwie pozycje według nie zmienionego porządku.

Jeśli więc pierwszym symbolem kodowanego strumienia danych będzie literka A, wtedy po dodaniu nowego symbolu drzewo będzie wyglądało jak na rys. 3.10.



Rys. 3.10. Adaptacyjne kodowe drzewo Huffmana po inicjalizacji i dodaniu nowego symbolu A, który znalazł się na początku kodowanego strumienia danych.

Następnie uruchamiany jest proces właściwej aktualizacji drzewa (według algorytmu 3.3), czyli zwiększenie wagi liścia symbolu A na 1 itd. Ze względu na konieczność dopisania nowych węzłów do listy wygodniej jest zmienić kolejność węzłów na odwrotną, czyli pierwszym na liście będzie korzeń drzewa, potem węzły o największych wagach aż do najmniejszych. Wówczas po dopisaniu dwóch nowych węzłów na końcu listy kolejność pozostałych się nie zmienia (jedynie ostatni węzeł otrzymuje o jeden większy indeks kosztem nowego węzła rodzica) i nie ma konieczności przesuwania wszystkich węzłów o dwie pozycje. Rozwiązania wymaga także problem wagi symbolu NEW. W przeciwieństwie do symbolu EOF, który wystąpi raz na końcu kodowania, a więc niezmiennie z wagą 1 będzie spadał na dół drzewa o rosnącym słowem kodowym, symbol NEW będzie emitowany tyle razy, ile nowych symboli pojawi się na wejściu. Można więc dla krótszych zbiorów inkrementować jego wagę po każdym wystąpieniu skracając długość przypisanego mu słowa. Inny sposób to pozostawienie jego wagi równej 1 jako stałą przewidując pojawianie się nowych symboli przede wszystkim na początku kodowania, gdy słowo symbolu NEW jest jeszcze krótkie. Oszczędzi to miejsce w wyższych partiach drzewa dla symboli danych, które częściej będą pojawiać się w strumieniu wejściowym.

Maksymalna liczba węzłów w drzewie Huffmana jest znana i jest równa podwojonej liczbie symboli minus jeden. Tak więc dla danych 8-mio bitowych i 256 możliwych symboli alfabetu liczba wszystkich węzłów nie przekroczy wartości 511. Nie jesteśmy jednak w stanie przewidzieć wag poszczególnych węzłów. W przykładowych strukturach węzłów prezentowanych w rozdz. 1 zdefiniowano do ich przechowywania element waga typu UINT. Przy kodowaniu dużych zbiorów danych może się okazać, że waga korzenia będąca sumą liczby danych, które przeszły przez koder plus wagi symboli pomocniczych (typu EOF i NEW) nie mieści się w tym elemencie pamięci. Potencjalnie liczba kodowanych znaków jest nieograniczona ...

Aby przystosować algorytm adaptacyjnej modyfikacji drzewa do rozwiązania takiego problemu, trzeba po pierwsze wykryć moment, kiedy możliwe jest przepełnienie komórek waga. Najprostszym rozwiązaniem wydaje się monitorowanie wagi korzenia. Gdy zbliża się ona do założonej granicznej wartości, wtedy trzeba się zdecydować na któryś z mechanizmów zaradczych. Można np. zamrozić w tym momencie postać drzewa, co jednak w przypadku zmiany charakteru danych w kolejnych partiach strumienia wejściowego znacznie zmniejsza skuteczność kodowania. Innym rozwiązaniem jest przeskalowanie drzewa np. poprzez podzielenie przez dwa wag węzłów zewnętrznych drzewa i konstrukcję nowego drzewa Huffmana. Taka modyfikacja może czasami dość znacząco zmienić postać drzewa, przy czym może to wpłynąć nawet korzystnie na skuteczność kodowania całego algorytmu. Czasami stosowane są bardziej wyrafinowane mechanizmy, jak chociażby zmniejszanie wag symboli,

które już dawno nie pojawiły się w strumieniu wejściowym lub nawet ich eliminacja (usunięcie odpowiadających im węzłów zewnętrznych).

W praktyce występuje jeszcze jeden istotny problem związany z przepełnieniem. Otóż w procedurze kodowania symbolu, który pojawił się na wejściu odszukiwany jest węzeł zewnętrzny opisujący ten symbol, a następnie rozpoczyna się podróż poprzez drzewo w kierunku korzenia, której towarzyszy odczytywanie kolejnych bitów kodu tego symbolu. Odczytany w ten sposób kod trzeba następnie odwrócić, tak że najbardziej znaczący bit opisuje wybór tej gałęzi wychodzącej z korzenia, która prowadzi do odpowiedniego liścia. Trzeba więc umieścić odczytywane słowa kodowe w komórce pamięci, co wymaga nałożenia kolejnego ograniczenia, w tym przypadku na maksymalną długość bitowego słowa przypisywanego poszczególnym symbolom alfabetu.

Aby globalnie rozwiązać problem przepełnienia, pomocnym może być ciąg Fibonacciego, definiowany przez poniższą formułę rekurencyjną:

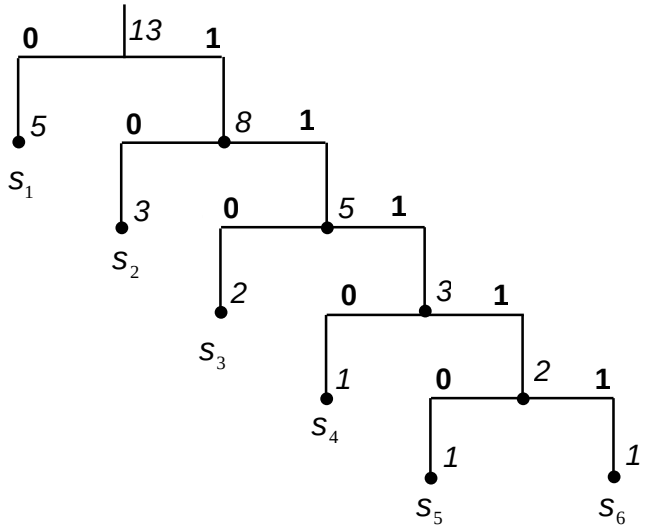
$$F(0)=0, \quad F(1)=1, \quad F(i)=F(i-1)+F(i-2) \quad \text{dla} \quad i>1. \tag{3.1}$$

Ciąg Fibonacciego wygląda więc następująco: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, ... Można wykazać, że jeśli waga korzenia drzewa Huffmana jest równa $F(i+2)$, to najdłuższa możliwa długość słowa kodowego przypisanego symbolowi na podstawie tego binarnego drzewa wynosi *i* bitów. Sytuacja, w której powstaje najdłuższe słowo kodowe w stosunku do wagi korzenia drzewa ma miejsce w przypadku rozrastającego się niesymetrycznie tzw. drzewa Fibonacciego, zbudowanego w przykładzie 3.3.

PRZYKŁAD 3.3. Ograniczenia przy adaptacyjnej rozbudowie drzewa.

Należy zbudować drzewo Huffmana dla alfabetu pięciu symboli o wagach wynikających ze statystyki wystąpień, które akurat ułożyły się w ciąg Fibonacciego (z dwukrotnym powtórzeniem pierwszego elementu ciągu), tj. $w_{S_1}=F(1)=1$, $w_{S_2}=F(1)=1$, $w_{S_3}=F(2)=1$, $w_{S_4}=F(3)=2$, $w_{S_5}=F(4)=3$, $w_{S_6}=F(5)=5$.

Drzewo takie wygląda następująco:



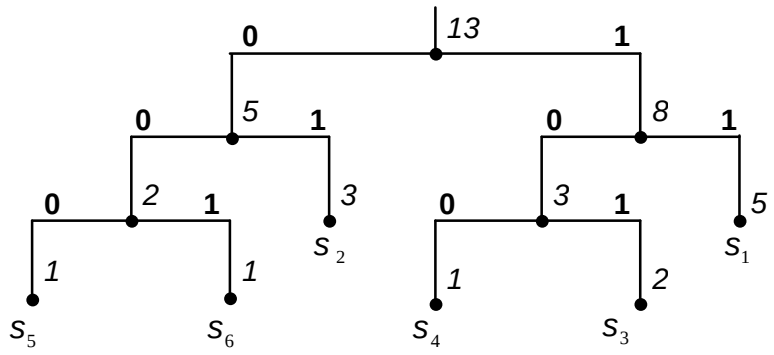
Rys. 3.11. Drzewo Fibonacciego. Wagi kolejnych węzłów są liczbami z ciągu Fibonacciego (z kilkoma powtórzeniami).

Na podstawie drzewa z rys. 3.11. możemy wykazać słuszność przedstawionej wyżej tezy o wskaźniku przepełnienia rejestrów słów kodowych drzewa Huffmana. Waga korzenia tego drzewa jest równa 13, czyli jest równa liczbie z ciągu Fibonacciego $F(7)=13$. Oznacza to, iż któremuś z symboli występujących w drzewie może zostać przypisane 5-cio bitowe słowo kodowe. Jak widać, zarówno s_5 jak i s_6 mają pięciobitowe słowa kodowe, odpowiednio 11110 i 11111. Nie sposób utworzyć dłuższego słowa kodowego w tym przypadku, gdyż liczba poziomów drzewa nie może przekroczyć pięciu (liczba węzłów zewnętrznych minus jeden). W takim drzewie wagi kolejnych węzłów drzewa są liczbami z ciągu Fibonacciego. Waga rodzica jest tworzona jak kolejny element ciągu Fibonacciego poprzez sumowanie wag dzieci – poprzednich elementów ciągu (patrz rys. 3.11). Waga węzła rodzica danego poziomu jest powtórzona na następnym wyższym poziomie w wadze kolejnego liścia, jako element $F(i-2)$. Z kolei waga rodzica na tym wyższym poziomie stanowi element $F(i-1)$ pozwalając wyznaczyć wagę ich wspólnego węzła rodzicielskiego na jeszcze wyższym poziomie, która jest równa kolejnemu elementowi ciągu $F(i)$.

W bardziej praktycznym przykładzie ograniczenia długości słowa kodowego wielkością 16 bitowego rejestru, wartość wagi korzenia, przy której musi nastąpić przeskalowanie drzewa wynosi $F(19)=4181$. Wtedy bowiem, w najbardziej niekorzystnym przypadku drzewa Fibonacciego, może pojawić się 17 bitowe słowo kodowe.

Algorytm 3.2 można nieco zmodyfikować realizując kod Huffmana, który jest bardziej przydatny w zastosowaniach transmisyjnych. Przy danej przepustowości kanału, chcąc zachować stałą prędkość transmisji w symbolach na sekundę lepiej jest, gdy poszczególne słowa kodowe różnią się możliwie najmniej. Punkt 3 algorytmu nic nie mówi o warunkach wyboru dwu węzłów o najmniejszych wagach w przypadku, gdy istnieje więcej węzłów z tą samą wagą należącą do zbioru dwu wag najmniejszych. Konstruując koder do celów transmisji można więc nałożyć dodatkowy warunek, że w pierwszej kolejności łączymy te z węzłów o równych wagach, które mają mniej węzłów potomnych (w danej chwili dołączone liście mają krótszy kod). Pozwala to uzyskać bardziej równomierne długości słów w ramach tego samego modelu statystycznego. Oczywiście, uzyskana efektywność kodowania nie zmieni się. Tak zmodyfikowany algorytm pozwala zrealizować kod Huffmana o minimalnej zmienności.

Prześledźmy powyższe dookreślenie algorytmu na przykładzie 3.3. Zmodyfikowane drzewo Huffmana przedstawiono na rys. 3.12.



Rys. 3.12. Drzewo Huffmana z przykładu 3.3 zmodyfikowane do celów transmisji danych kodowanych, pozwalające zrealizować kod Huffmana o minimalnej zmienności.

Dla drzewa z rys. 3.11 długości kodów przypisanych kolejnym symbolom wynoszą odpowiednio 1, 2, 3, 4, 5 i 5 bitów. Daje to całkowitą długość kodu (suma iloczynów liczby wystąpień i długości słowa kodowego dla wszystkich symboli) wynoszącą 31 bitów przy silnym zróżnicowaniu długości poszczególnych słów kodowych. Z drzewa na rys. 3.12 wynika, że długość słów kolejnych symboli jest następująca: 2, 2, 3, 3, 3 i 3 bity, co daje tę samą długość strumienia wyjściowego 31 bitów przy w przybliżeniu równomiernej długości słów. Widać także, że w tym przypadku znacznie łagodniejsze mogą być ograniczenia dotyczące przepełnień rejestrów ze słowami kodowymi.

Większe koszty obliczeniowe wersji adaptacyjnej kodera Huffmana, jak również ograniczona zdolność adaptacji, która nie zawsze może nadążyć za zmieniającą się lokalnie charakterystyką powoduje, że metoda ta daje zazwyczaj wyraźnie lepsze rezultaty w porównaniu do metody statycznej jedynie w przypadku kompresji typowych (naturalnych) zbiorów danych znacznych rozmiarów, o których wiemy bardzo niewiele.

3.4. Znormalizowany statyczny koder Huffmana

Adaptacyjny koder Huffmana może być zastąpiony szybszym i równie skutecznym algorytmem statycznym jedynie w przypadku odpowiednio bogatej wiedzy dostępnej *a priori* na temat kompresowanego zbioru danych. Może to być znany rozkład wartości błędu predykcji, znana statystyka źródła informacji przy kodowaniu wielu zbiorów o tym samym charakterze danych, np. zawierające wartości dobowe odczytu temperatury oraz ciśnienia z kilku czujników, lub inne.

Jako przykład użytecznego kodera Huffmana w wersji statycznej rozważmy schemat kodowania kwantowanych współczynników DCT (transformaty kosinusowej – patrz punkt 11.1) w podstawowym schemacie standardu kompresji obrazów JPEG. Zastosowano tutaj statyczną metodę Huffmana w połączeniu z RLE, której konstrukcję oparto na znanej średniej charakterystyce wartości współczynników transformaty kosinusowej (malejące wartości danych w strumieniu uformowanym do kodowania, z dużą ilością zer na końcu niemal każdego bloku 64 współczynników).

Mechanizm RLE, jak również kodowania Huffmana został wbudowany w hybrydowy algorytm kodowania kwantowanych wartości współczynników. Wykorzystaną wiedzę *a priori* można sprowadzić zasadniczo do ustalenia, że małe wartości współczynników występują znacznie częściej niż duże, wobec czego długość słów kodowych im przypisanych powinna być wyraźnie krótsza. Ponadto, z faktu występowania dużej liczby zer w blokach współczynników po kwantyzacji wynikło włączenie powtarzających się zer w mechanizm budowania słów kodu Huffmana dla współczynników niezerowych. Odbyło się to na podstawie analizy częstości ich występowania w sąsiedztwie małych i dużych wartości współczynników DCT oraz na końcu sekwencji współczynników danego bloku, uporządkowanych metodą zygzak (patrz punkt 11.4). Ze względu na odmienną charakterystykę ustalono różny sposób kodowania składowej stałej (DC) i harmonicznych (AC) transformaty kosinusowej. Wyjściowe słowo kodowe współczynników AC konstruowane jest z trzech zasadniczych części: liczby zer poprzedzających daną niezerową wartość kodowanego współczynnika (to jest właśnie składowa RLE) z danego bloku, kategorii określającej przedział wartości do której trafia dany współczynnik oraz dookreślenia wartości współczynnika wewnątrz danej kategorii. Liczba bitów dookreślających wartość współczynnika jest jednocześnie numerem kategorii. Innymi słowy RLE położenia w bloku w połączeniu z kategoryzacją wartości współczynnika zostały opisane kodem Huffmana i uzupełnione klasycznym zapisem w notacji binarnej.

W przypadku składowej stałej, kodowana wartość współczynnika jest korygowana różnicowo przy pomocy prostej predykcji: od wartości współczynnika DC danego bloku odejmowana jest wartość DC bloku poprzedzającego w sekwencji skanowania bloków danego obrazu. Wykorzystuje się w ten sposób korelację składowej stałej sąsiednich bloków. Wartość różnicowa klasyfikowana jest to jednej z 12 kategorii, w tym również zerowej kategorii przypisanej zerowej wartości różnicowej. Te zerowe wartości kodowane są jedynie przy pomocy słowa kodu Huffmana ustalonego dla kategorii 0. Słowa kodowe przypisane niezerowym wartościom różnicowym zbudowane są ze sklejonych ze sobą dwu części: słowa kodu Huffmana przydzielonego danej kategorii (o długości dobranej na odstawie średniej statystyki wystąpień wartości danej kategorii), np. zgodnie z tabelą 3.4, oraz określonej przez kategorię (pierwsza kategoria – jeden bit, druga – dwa, itd.) liczby mniej znaczących bitów wartości współczynnika *C* (dla dodatnich *C*) lub liczby mniej znaczących bitów wartości *C*-1 (dla współczynników ujemnych). Zakłada się, że wszystkie wartości danej kategorii są jednakowo prawdopodobne, stąd przydziela się im słowa o tej samej długości, które uzupełniają informację przekazaną dekoderowi przez kod Huffmana kategorii.

Tabela 3.4. Klasyfikacja współczynników DC transformaty kosinusowej ze względu na wartość, w celu efektywnego ich kodowania. Obok kategorii i długości słowa kodowego podano również typową postać słów kodowych luminancji (nie jest obligatoryjna w standardzie).

Kategoria	Zakres wartości różnicowej dla współczynnika DC	Długość słowa kodowego	Słowo kodowe
0	0	2	00
1	-1,1	3	010
2	-3,-2,2,3	3	011
3	-7,...,-4,4,...,7	3	100
4	-15,...,-8,8,...,15	3	101
5	-31,...,-16,16,...,31	3	110
6	-63,...,-32,32,...,63	4	1110
7	-127,...,-64,64,...,127	5	11110
8	-255,...,-128,128,...,255	6	111110
9	-511,...,-256,256,...,511	7	1111110
10	-1023,...,-512,512,...,1023	8	11111110
11	-2047,...,-1024,1024,...,2047	9	111111110

Współczynniki AC, kodowane inaczej niż składowa stała DC, ze względu na niezerową amplitudę przyporządkowane są do jednej z dziesięciu kategorii przedstawionych w tabeli 3.5. Aby zakodować współczynniki AC, każda niezerowa wartość współczynnika jest opisana w pierwszej kolejności przez 8-mio bitową wartość w postaci binarnej: RS='RRRRSSSS', która definiuje alfabet dla kodu Huffmana. Cztery mniej znaczące bity 'SSSS' określają kategorię, do której należy niezerowa wartość współczynnika według tabeli 3.5. Cztery bardziej znaczące bity 'RRRR' opisują pozycję współczynnika względem poprzedniego niezerowego współczynnika (tj. liczbę oddzielających je zerowych współczynników) w kolejności ustalonej według sekwencji zygzak. Jeśli liczba zer dzielących dwa niezerowe

współczynniki przekracza 15, wówczas definiowana jest wartość: RS='F0' (heksadecymalnie) interpretowana jako ciąg 15 zer, po którym występuje wartość zerowa (czyli przy dekodowaniu ten symbol zostanie zastąpiony przez 16 zerowych wartości kolejnych współczynników). Ponadto, ustala się dodatkowy symbol postaci RS='00' (heksadecymalnie) określający koniec wartości współczynników danego bloku (EOB). Znaczy to, że wszystkie pozostałe współczynniki bloku (od tego miejsca w kolejności według sekwencji zygzak) są zerowe.

Tabela 3.5. Klasyfikacja współczynników AC transformaty kosinusowej ze względu na wartość, w celu efektywnego ich kodowania.

Kategoria (liczba bitów uzupełniającej części słowa kodowego)	Zakres wartości współczynnika AC
1	-1,1
2	-3,-2,2,3
3	-7,...,-4,4,...,7
4	-15,...,-8,8,...,15
5	-31,...,-16,16,...,31
6	-63,...,-32,32,...,63
7	-127,...,-64,64,...,127
8	-255,...,-128,128,...,255
9	-511,...,-256,256,...,511
10	-1023,...,-512,512,...,1023

Słowa kodowe Huffmana (patrz przykładowa tabela 3.6) przypisane wartościom RS, które są wynikiem złożenia opisu względnej pozycji współczynnika metodą RLE oraz kategoryzacji wartości współczynników niezerowych, mają długość od 2 do 16 bitów. Słowa te są uzupełniane bitami dookreślającymi wartość współczynnika w sposób następujący. Ponieważ wartości kategorii *k*='SSSS' należą do przedziału $\left(2^{k-1},2^{k}-1\right)$ lub $\left(-2^{k}+1,-2^{k-1}\right)$, gdzie $1 \le k \le 10$, to *k* mniej znaczących bitów wartości współczynnika *C* stanowi dookreślającą część dla dodatnich *C*, podczas gdy *k* mniej znaczących bitów wartości *C*-1 stanowi dookreślającą część dla współczynników ujemnych. Zakłada się, że wszystkie wartości danej kategorii są jednakowo prawdopodobne, stąd przydziela się im słowa (uzupełniające) w reprezentacji binarnej o tej samej długości (analogicznie jak przy kodowaniu składowej stałej). Przykładowo, korzystając z tablicy kodowania przedstawionej w tabeli 3.6 słowa kodowe dla ciągu wartości współczynników DCT: ..., niezerowy, 0, -3, 0, 0, 0, 1, -7, ... ustalane są następująco (zaczynając od współczynnika równego –3): ponieważ wartości słów RS dla trzech niezerowych współczynników z ciągu są równe odpowiednio: 1/2, 3/1, 0/3, to po odczytaniu ich kodu Huffmana z tablicy 3.6 oraz uzupełnieniu ich dokładnymi wartościami współczynników danej kategorii uzyskuje się ostateczne słowa kodowe postaci: 11100100, 1110101, 100000.

Tabela 3.6. Przykładowe słowa kodowe luminancji przypisane współczynnikom AC w typowej tablicy standardu JPEG (nie obligatoryjnej).

Słowo RS (liczba poprzedzających zer/ kategoria)	Długość słowa kodowego	Słowo kodowe
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
...	…	…
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	6	111001
....	…	…
2/A	16	1111111110001110
3/1	6	111010
....	…	…
C/9	16	1111111111100000
C/A	16	1111111111100001
D/1	11	11111111000
...	...	...
F/0 (same zerowe współczynniki)	11	11111111001
F/1	16	1111111111110101
...	…	…
F/9	16	1111111111111101
F/A	16	1111111111111110

3.5. Podsumowanie

W metodzie Huffmana, ze względu na ograniczenia związane z oddzielnym przydzielaniem słów kodowych (o całkowitej liczbie bitów) dla każdego symbolu, o długości odwrotnie proporcjonalnej do jego prawdopodobieństwa, nie stosuje się praktycznie modeli źródeł wyższego rzędu. Proces kompresji ograniczony jest więc do prostego modelowania źródła informacji bez pamięci z częstościową estymacją prawdopodobieństw oraz przydziałem różnej długości bitowych słów kodowych poszczególnym symbolom alfabetu w etapie binarnego kodowania. Dynamiczne modelowanie zawartości alfabetu i prawdopodobieństw symboli pojawia się w adaptacyjnej wersji algorytmu, kiedy to statystyka decydująca o postaci słowa kodowego przydzielanego poszczególnym symbolom źródła zmieniana jest lokalnie, a więc model opisujący dane wierniej je charakteryzuje pozwalając zwiększyć skuteczność kompresji. Widać to w wynikach eksperymentów przeprowadzonych na kilku zbiorach tekstowych kodowanych statycznym i adaptacyjnym koderem Huffmana, pokazanych w tabeli 3.7. Uzyskano krótszą o blisko 0.4 bita/symbol reprezentację zbiorów dla rozwiązania dynamicznego, bez względu na dość zróżnicowaną długość plików i rodzaj danych.

Tabela 3.7. Porównanie efektywności kompresji realizacji kodera Huffmana w wersji statycznej i adaptacyjnej. Zbiory Z1-Z3 to zbiory tekstowe o różnych rozmiarach. Tabela zawiera wartości średnich bitowych kodu skompresowanych zbiorów w bitach na daną.

Koder	Zbiór			Średnio
	Z1 10kB	Z2 100kB	Z3 4000kB	
Huffman statyczny	5.85	8.06	2.43	5.45
Huffman adaptacyjny	5.33	7.78	2.10	5.07

Koder Huffmana znajduje częste zastosowanie w dzisiejszych systemach kompresji i archiwizacji danych. Nie najlepsza skuteczność kodowania samego algorytmu Huffmana wymaga uzupełnienia schematu kompresji najczęściej poprzez skonstruowanie bardziej efektywnej wstępnej metody modelowania danych. Tak więc metodę Huffmana wykorzystuje się z dużym powodzeniem do kodowania reszt predykcyjnych, których rozkład wartości jest w przybliżeniu znany (rozkład Laplace'a o zerowej wartości średniej i parametryzowanej wariancji). Metody słownikowe (szczególnie LZ78) o stałym rozmiarze słownika dają na wyjściu ciąg indeksów do słownika, które można zakodować wykorzystując statystyczne modele adaptacyjnego kodera Huffmana.

Bibliografia

1. Gary Stix, *Profile: David Huffman,* Scientific American, September 1991.
2. D. A. Huffman, *A Method for the Construction of Minimum Redundancy Codes*, Proceedings of the IRE, 40:1098-1101, 1952.
3. N. Faller, *An Adaptive System for Data Compression,* Record of the 7th Asilomar Conference on Circuits, Systems, and Computers, IEEE Press, 593-597, 1973.
4. R. G. Gallagher, *Variations on a Theme by Huffman,* IEEE Trans. Inform. Theory, 24(6):668-674, 1978.
5. D. E. Knuth, *Dynamic Huffman Coding,* Journal of Algorithms, 6:163-180, 1985.

6. J. S. Vitter, *Design and Analysis of Dynamic Huffman Codes,* Journal of ACM, 34(4):825-845, 1987.
7. X. Lin, *Dynamic Huffman Coding for Image Compression,* praca magisterska w University of Nebraska, 1991.
8. M. Nelson, *The Data Compression Book*, M&T Books, rozdz. 3 i 4, 1991.