1.4.3. Kanoniczne kodowanie Huffmana

Kanoniczne kodowanie Huffmana zostało po raz pierwszy opisane przez Schwartza i Kallicka w 1964 r. [42]. Jest to pewne ujednoznacznienie kodowania Huffmana, ponieważ dla danego rozkładu prawdopodobieństwa symboli zawsze istnieje dokładnie jedno drzewo kodowe. Dekodowanie tekstu jest szybsze w porównaniu z podstawowym kodowaniem Huffmana. Dekoderowi nie jest potrzebna znajomość całego drzewa kodowego, a wystarczy tylko liczba wierzchołków na kolejnych poziomach drzewa.

Kody Huffmana są powszechnie stosowane, więc ważna jest efektywność ich implementacji. Mimo tego, że kanoniczne kody Huffmana stanowią najbardziej wydajną implementacją kodów Huffmana, to rzadko można spotkać ich opis w podręcznikach. Z tego powodu omówimy je dość dokładnie, po szczegóły odsyłając do pracy [42].

Algorytm Huffmana (Algorytm 2) nie precyzuje, które poddrzewa należy połączyć w sytuacji, gdy jest więcej możliwości — gdy są co najmniej trzy poddrzewa o minimalnej częstości lub jest jedno o minimalnej częstości, ale więcej niż jedno poddrzewo o częstości drugiej co do wielkości.

W kroku 4 Algorytmu 2 wybieramy parę najmniejszych drzew z listy, z których tworzymy nowe drzewo. Wybrane drzewa usuwamy z listy, a nowo powstałe drzewo wstawiamy do niej według nadanej częstości. To wstawienie ma wpływ na kształt drzewa. Kanoniczne kodowanie Huffmana polega na wybraniu drzew najmniejszych w sensie innego porządku na drzewach. Według porządku $\preceq_{kan}$ drzewa o tej samej częstości (zwykle) będą rozróżniane. Dzięki temu dla takiego samego rozkładu prawdopodobieństwa otrzymamy drzewo o takich samych własnościach (takim samym kształcie).

Definicja 1.3 (Porządek $\preceq_{kan}$)**.** Niech czas utworzenia drzewa t będzie liczbą kroków Algorytmu 2 wykonanych do momentu utworzenia drzewa t i przyjmujemy, że wszystkie drzewa jednowierzchołkowe z kroku 1 tego algorytmu są tworzone w tym samym czasie. Niech t_1 i t_2 będą drzewami o częstościach odpowiednio f_1, f_2, utworzonymi w czasie τ_1, τ_2. Porządek na drzewach kodowych definiujemy następująco:

$$t_1 \preceq_{kan} t_2 \iff f_1 < f_2 \vee (f_1 = f_2 \wedge \tau_1 \leqslant \tau_2)\,.$$

Nowe drzewo t o częstości f (tworzone w kroku 4 Algorytmu 2) jest większe od wszystkich utworzonych do tej pory drzew o tej częstości. Podczas wstawiania t do listy drzew nie zawiera ona drzewa t', dla którego zachodziłby warunek:

$$t \neq t' \wedge t \preceq_{kan} t' \wedge t' \preceq_{kan} t\,.$$

Wobec tego zawsze kształt dwóch najmniejszych drzew według porządku $\preceq_{kan}$ jest ściśle określony. Mówimy o kształcie, ponieważ drzewa jednowierzchołkowe nie są rozróżniane, wybór nie jest jednoznaczny, jednak nie ma wpływu na własności drzewa.

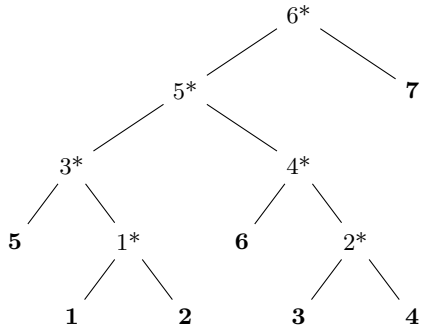
Algorytm 3 Budowanie kanonicznego drzewa kodowego Huffmana

1: Utwórz listę L drzew jednowierzchołkowych przechowujących parę $\langle s_i, f_i \rangle$, dla każdego symbolu s_i o częstości f_i. Częstością takiego drzewa jest f_i.
2: **while** na liście L jest więcej niż jedno drzewo **do**
3: Usuń z listy L dwa najmniejsze w sensie relacji $\preceq_{kan}$ drzewa t_0 i t_1 o częstościach odpowiednio: f_0, f_1.
4: Utwórz nowe drzewo t, którego częstością jest $f = f_0 + f_1$, a drzewa t_0 i t_1 są odpowiednio jego lewym i prawym poddrzewem.
5: Wstaw drzewo t do listy L.
6: **end while**
7: **if** lista L nie jest pusta **then**
8: Znormalizuj jedyne drzewo listy L
9: **end if**

Kodowanie drzewa

Celem pierwszego etapu kodowania jest ustalenie liczby liści na każdym poziomie drzewa kodowego. Skonstruujemy drzewo kodowe, które będzie miało najmniejszą możliwą wysokość spośród wszystkich możliwych drzew kodowych Huffmana. Ta własność jest o tyle ważna, że naszym celem jest zakodowanie drzewa w taki sposób, że rozmiar zakodowanej postaci jest proporcjonalny do wysokości drzewa kodowego. Algorytm 3 stanowi zapis algorytmu kanonicznego kodowania Huffmana.

Przykład 1.1. Poniżej przedstawione są listy drzew w kolejnych krokach algorytmu, t oznacza nazwę drzewa, f jego częstość. Pierwsza lista zawiera tylko drzewa jednowierzchołkowe. Drzewo utworzone w krokach 1 – 6 Algorytmu 3 dla podanych częstości występowania symboli w tekście znajduje się na Rysunku 1.9.



Rysunek 1.9: Drzewo kodowe utworzone według Algorytmu 3 przed normalizacją

t	f
1	1
2	1
3	1
4	2
5	2
6	3
7	12

t	f
3	1
4	2
5	2
1*	2
6	3
7	12

t	f
5	2
1*	2
6	3
2*	3
7	12

t	f
6	3
2*	3
3*	4
7	12

t	f
3*	4
4*	6
7	12

t	f
5*	10
7	12

t	f
6*	22

◇

Chcemy efektywnie zakodować drzewo kodowe. Aby umożliwić zakodowanie drzewa w pamięci proporcjonalnej do jego wysokości, potrzebna jest normalizacja. Stąd ostatni krok Algorytmu 3 stanowi normalizacja drzewa kodowego.

Pomocniczą strukturą danych, która umożliwi odtworzenie znormalizowanego drzewa kodowego, będzie lista *levels*, taka że dla $i \in [1\,,\,N]$ wartość $levels[i]$ oznacza liczbę liści na poziomie i. Drzewo utworzone w krokach $1-6$ Algorytmu 3 będzie modyfikowane jeszcze, aby ta lista wystarczyła do późniejszego odtworzenia drzewa kodowego.

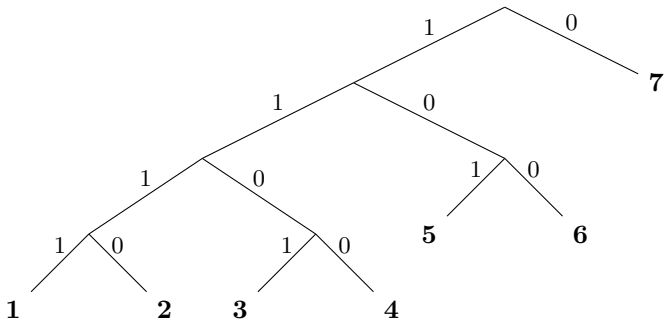
Kolejne elementy listy odpowiadają kolejnym poziomom od pierwszego, czyli liściom od najmniejszej głębokości. Dla rozważanego tu przykładu $levels = [1, 0, 2, 4]$.

Zanim przejdziemy do nadania słów kodowych symbolom na podstawie utworzonej listy *levels*, zmodyfikujemy drzewo kodowe, aby łatwo było nadać słowa kodowe symbolom. Modyfikację nazywać będziemy normalizacją z tego względu, że lista *levels* dla zmodyfikowanego drzewa pozostanie niezmieniona.

Normalizacja będzie polegała na przesunięciu liści tak, by po prawej stronie każdego liścia nie było wierzchołków wewnętrznych i w taki sposób, żeby po tej mo-

dyfikacji wszystkie liście były w tej samej kolejności na swoim poziomie co w oryginalnym drzewie. Sytuację tę obrazuje Rysunek 1.10. Zauważmy, że lista *levels* stanowi reprezentację znormalizowanego drzewa kodowego, ponieważ istnieje tylko jedno znormalizowane drzewo, któremu odpowiada dana lista *levels*.

Każdy symbol otrzymuje słowo kodowe według tak otrzymanego drzewa w następujący sposób. Słowo kodowe jest budowane przy przechodzeniu od korzenia do liścia przechowującego symbol. Przy przejściu do lewego syna dodajemy do słowa kodowego 1, a przy przejściu do prawego — 0.



Rysunek 1.10: Drzewo kodowe kanonicznego kodowania Huffmana, zmodyfikowane drzewo z Rysunku 1.9

Pokażemy, jak można nadać słowa kodowe symbolom wyłącznie na podstawie listy *levels*. Słowa kodowe będą nadawane liściom w malejącym porządku ich częstości. Niech i_0 będzie najmniejszym indeksem, dla którego $levels[i_0] \neq 0$. Liść o największym numerze znajduje się na głębokości i_0 i otrzymuje słowo kodowe złożone z i_0 zer. Następnie nadawane są słowa kodowe wszystkim liściom, które znajdują się na tej samej głębokości, jeśli takie istnieją. Kolejno każdy liść otrzymuje słowo kodowe, który powstaje przez binarne dodanie 1. Po nadaniu słów kodowych wszystkim liściom o głębokości i, nadajemy je następnym liściom na głębokości i_1, gdzie i_1 jest minimalną wartością taką, że $i_1 > i$ oraz $levels[i_1] \neq 0$. Pierwszy liść na głębokości i_1 otrzymuje słowo kodowe powstałe ze słowa kodowego poprzedniego liścia (ostatniego na poziomie i) przez binarne dodanie 1 i dopisanie na końcu $i_1 - i$ zer. Algorytm 4 jest zapisem tej procedury.

Przykład 1.2. W poniższej tabeli podaliśmy słowa kodowe dla wszystkich liści drzewa kodowego, którego lista *levels* to $[1, 0, 2, 4]$.

Algorytm 4 Nadawanie słów kodowych liściom

Wejście: lista *levels*; liczba liści

Wyjście: tablica kodów *code* taka, że $code[i]$ jest kodem liścia o numerze i

```
                                          ▷ x ⊕ '0' oznacza dopisanie na końcu znak '0'
 1: CurrCode ← '0'
 2: leaf ← liczba liści                   ▷ zaczynamy od liścia o największym numerze
 3: l ← 1
 4: while l ⩽ |levels| do
 5:     while levels[l] = 0 do                        ▷ dopisanie zer na końcu kodu
 6:         CurrCode ← CurrCode ⊕ '0'
 7:         l ← l + 1
 8:     end while
 9:     code[leaf] ← CurrCode
10:     for i ← 2 to levels[l] do     ▷ nadawanie kodów liściom na jednym poziomie
11:         code[leaf − 1] ← code[leaf] + 1
12:         leaf ← leaf − 1
13:     end for
14:     l ← l + 1                                   ▷ przejście do kolejnego poziomu
15:     CurrCode ← code[leaf] + 1 ⊕ '0'             ▷ początek kodu pierwszego liścia
    z kolejnego poziomu
16:     leaf ← leaf − 1
17: end while
```

liść	kod
7	0
6	1 **0 0**
5	1 0 1
4	1 1 0 **0**
3	1 1 0 1
2	1 1 1 0
1	1 1 1 1

◇

Zauważmy, że liść o największym numerze zawsze otrzymuje słowo kodowe złożone z samych zer a liść o najmniejszym numerze słowo kodowe złożone z samych jedynek.

Dekodowanie

Zadaniem dekodera jest odtworzenie słów kodowych symboli na podstawie liczby wierzchołków na kolejnych poziomach drzewa kodowego. Przy ustalaniu kodów dla wierzchołków odpowiadających symbolom korzystaliśmy tylko z listy *levels*, toteż przy dekodowaniu słowa kodowe można ustalić dokładnie w ten sam sposób.

Na podstawie listy *levels* zbudujemy nową listę $code_{\max}$, która pozwoli na dekodowanie tekstu zakodowanego kanonicznym kodowaniem Huffmana. Lista $code_{\max}$ będzie przechowywać informację dotyczącą wierzchołka v_l, który jest liściem o najmniejszym numerze na danym poziomie l. Element $code_{\max}[l]$ jest słowem kodowym wierzchołka v_l. Jeżeli na poziomie l nie ma żadnego liścia, to $code_{\max}[l] = 0$. Zauważmy, że liście z poziomu i otrzymują słowa kodowe od $code_{\max}[i] - levels[i] + 1$ do $code_{\max}[i]$.

Słowa kodowe w liście $code_{\max}$ będą przechowywane jako liczby w zapisie binarnym. Mimo, że wiele (nawet nieskończenie wiele) słów kodowych mogłoby mieć taki sam zapis binarny ze względu na pomijanie początkowych zer, to dla ustalonej długości słowa kodowego będzie można je zawsze ustalić. Dlatego dalej będziemy utożsamiać słowo kodowe z jego zapisem binarnym.

Przykład 1.3. Dla listy $levels = [1, 0, 2, 4]$ otrzymujemy listę $code_{\max}$

poziom:	1	2	3	4
$code_{\max}$	0	0	101	1111

◇

Proces dekodowania polega na odczytywaniu słowa kodowego bit po bicie. W każdym momencie aktualne słowo kodowe jest porównywane z elementem $code_{\max}[l]$, gdzie l jest liczbą przeczytanych bitów słowa kodowego. Gdy $code_{\max}[l]$ okaże się mniejszy, to znaczy, że słowo kodowe jest dłuższe, więc należy przeczytać kolejny bit. Te kroki powtarzamy aż do momentu, w którym wartość liczbowa przeczytanego słowa kodowego nie jest większa $code_{\max}[l]$.

1.4.4. Własności kodów Huffmana

Kody Huffmana są prefiksowe i optymalne spośród wszystkich kodów prefiksowych. Kanoniczne kody Huffmana, oprócz tej cechy, charakteryzują się także najmniejszą:

- długością najdłuższego słowa kodowego [42],
- sumaryczną długością słów kodowych [42, 41],
- wariancją długości słów kodowych [42, 41].

Uzyskanie kodów o najmniejszej możliwej długości najdłuższego słowa kodowego jest ważne z tego względu, że ma wpływ na długość listy *levels*, używanej do wyznaczenia słów kodowych.