4d. Algorytm Huffmana

Grzegorz Kosiorowski

Uniwersytet Ekonomiczny w Krakowie

Drzewo z wagami

Drzewo z wagami

Drzewem z wagami nazywamy skończone drzewo z wyróżnionym korzeniem, w którym każdemu liściowi jest przyporządkowana nieujemna liczba rzeczywista zwana **wagą liścia**.

Drzewo z wagami

Drzewo z wagami

Drzewem z wagami nazywamy skończone drzewo z wyróżnionym korzeniem, w którym każdemu liściowi jest przyporządkowana nieujemna liczba rzeczywista zwana **wagą liścia**.

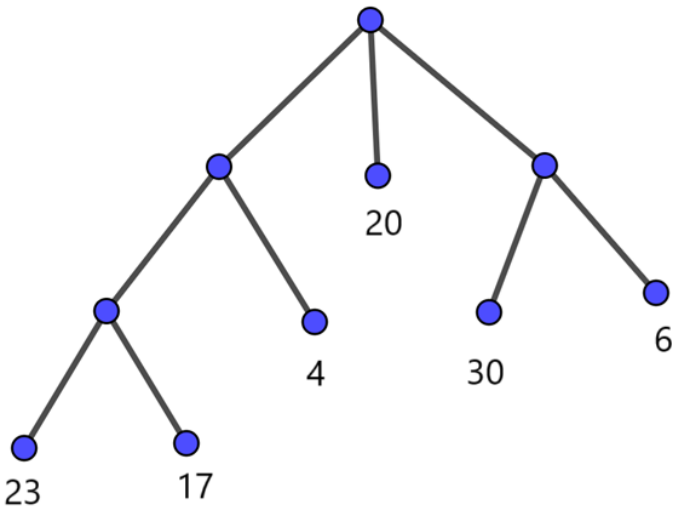
Waga drzewa

Wagą drzewa z wagami T, które ma n liści o wagach $w_1, w_2, \ldots w_n$, których poziomy (odległości od korzenia) oznaczamy odpowiednio $l_1, l_2, \ldots, l_n$ nazywamy liczbę:

$$W(T) = \sum_{i=1}^{n} l_i w_i.$$

Wagi drzew - przykład

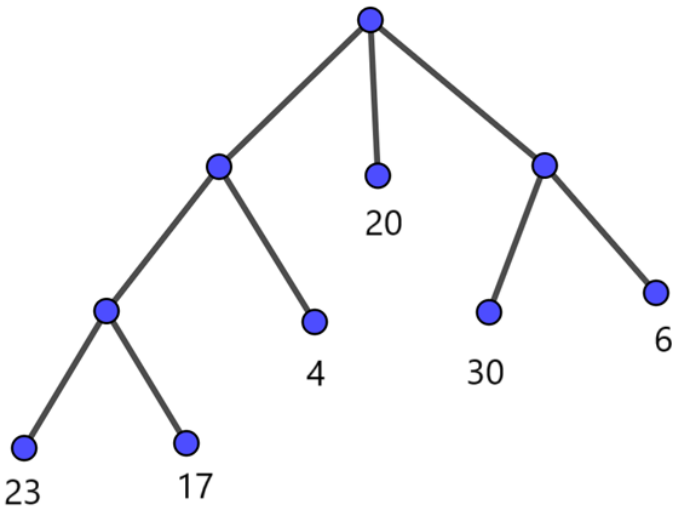
Przykładowe drzewa z wagami liści: 30, 23, 20, 17, 6 i 4:

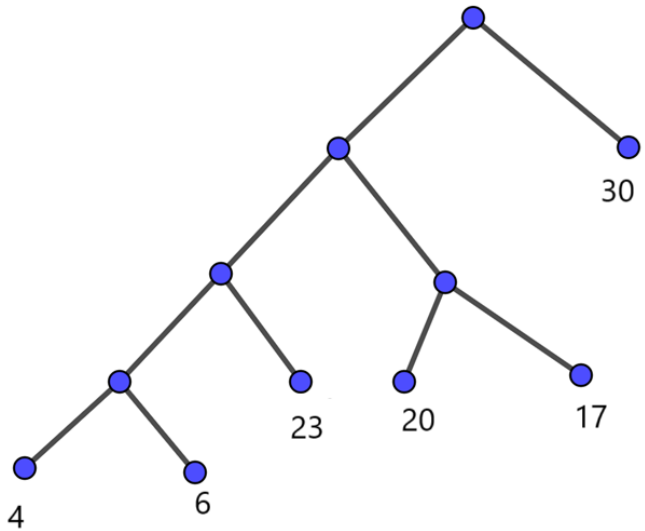


$$W(T) = 1 \cdot 20 + 2 \cdot (4 + 30 + 6) +$$
$$+ 3 \cdot (23 + 17) = 220.$$

Wagi drzew - przykład

Przykładowe drzewa z wagami liści: 30, 23, 20, 17, 6 i 4:



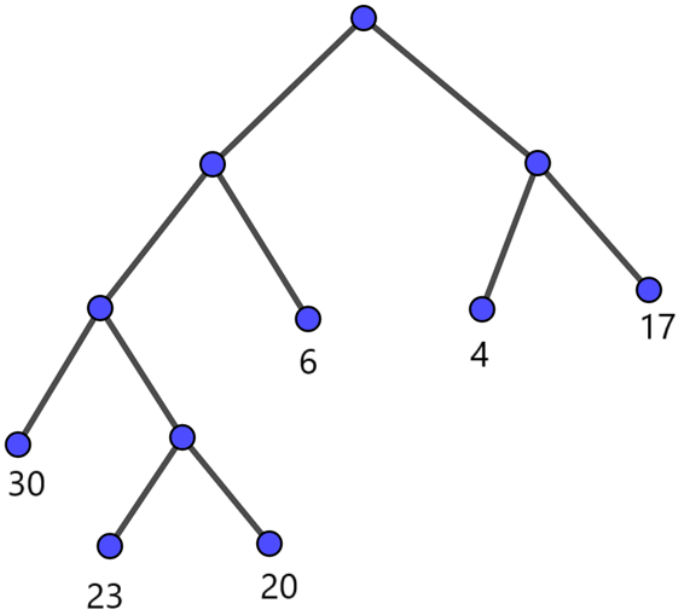


$$W(T) = 1 \cdot 20 + 2 \cdot (4 + 30 + 6) + \\ + 3 \cdot (23 + 17) = 220.$$

$$W(T) = 1 \cdot 30 + 3 \cdot (23 + 20 + 17) + \\ + 4 \cdot (4 + 6) = 250.$$

Wagi drzew - przykład

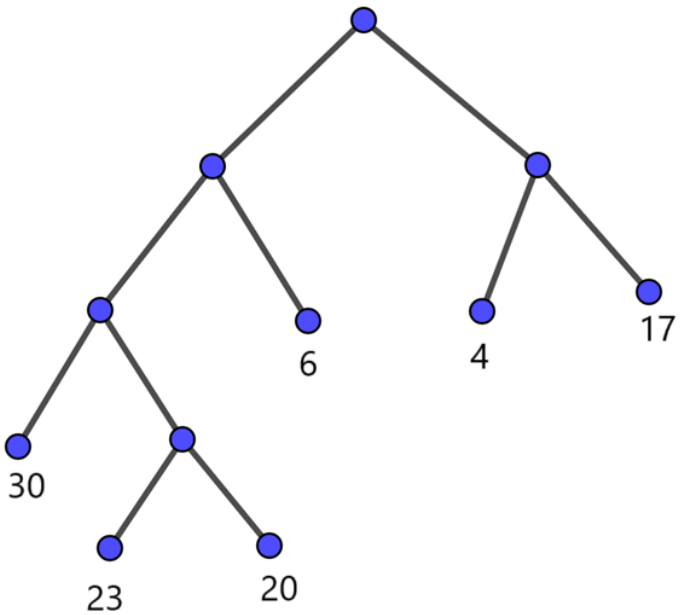
Przykładowe drzewa z wagami liści: 30, 23, 20, 17, 6 i 4:

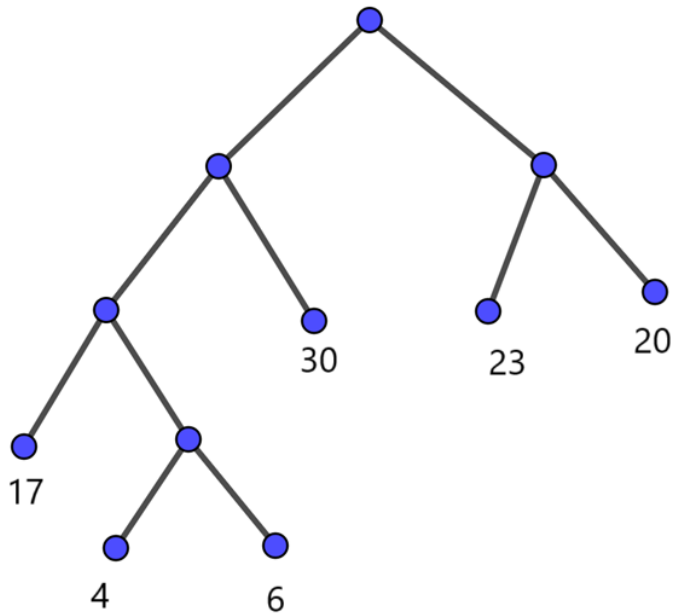


$$W(T) = 2\cdot(6+4+17)+3\cdot 30+$$
$$+4\cdot(23+20) = 316.$$

Wagi drzew - przykład

Przykładowe drzewa z wagami liści: 30, 23, 20, 17, 6 i 4:





$$W(T) = 2\cdot(6+4+17)+3\cdot 30+ \\ +4\cdot(23+20) = 316.$$

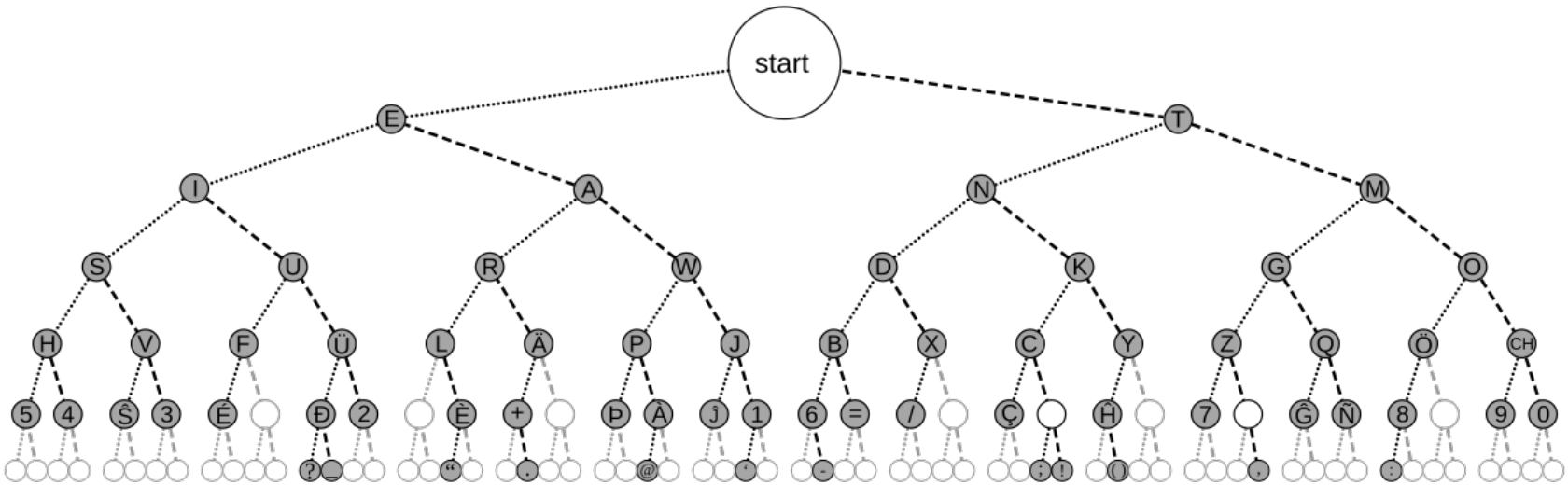
$$W(T) = 2\cdot(30+23+20)+3\cdot 17+ \\ +4\cdot(4+6) = 237.$$

Wagi drzew: minimalizacja

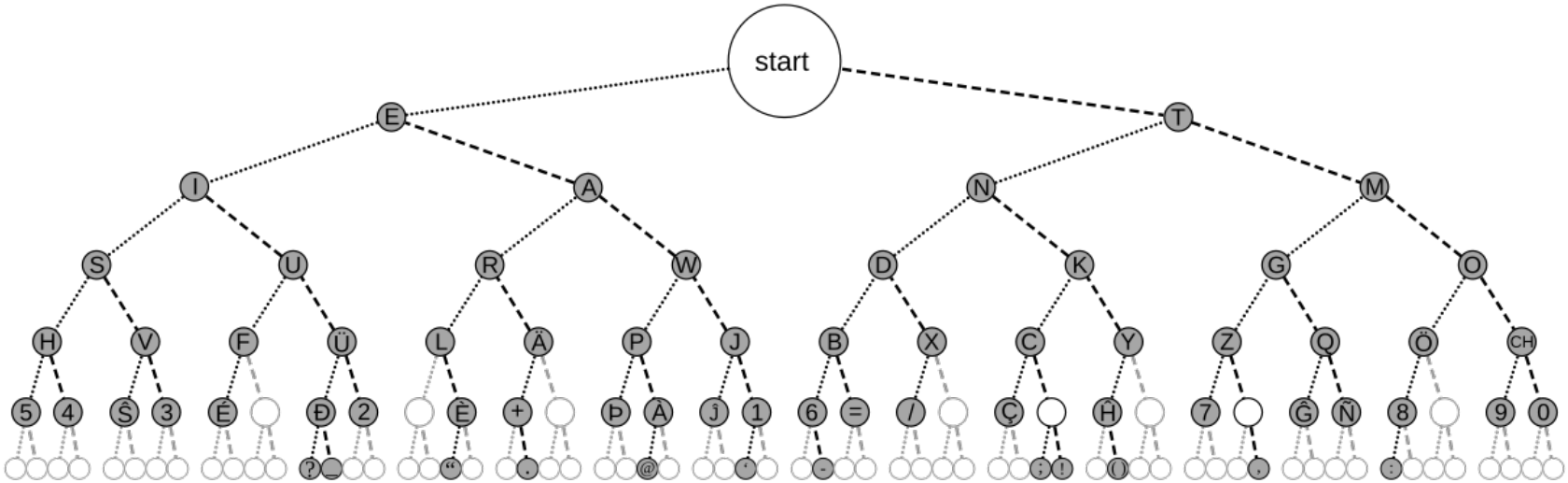
Jak widać, nawet izomorficzne drzewa o takich samych, lecz inaczej rozmieszczonych wagach liści, mogą mieć różne wagi.
Szczególne znaczenie w wielu praktycznych zagadnieniach ma wyznaczanie drzewa binarnego z liśćmi o zadanych wagach, które samo ma jak najmniejszą wagę. Wagę minimalną takiego drzewa (i samo drzewo o tej wadze) wyznacza tak zwany algorytm Huffmana.

Konwersje na format cyfrowy

We wszystkich zagadnieniach konwersji na format cyfrowy pojawia się pytanie: jak najefektywniej zamienić symbole danego języka na zapis binarny? Pierwszą słynną próbą takiej zamiany jest alfabet Morse'a, który wszystkie znaki alfabetu (przynajmniej angielskiego) zamienia na ciąg kresek i kropek (co jest równoważne z zamianą na ciąg 1 i 0). Obrazek z Wikipedii:

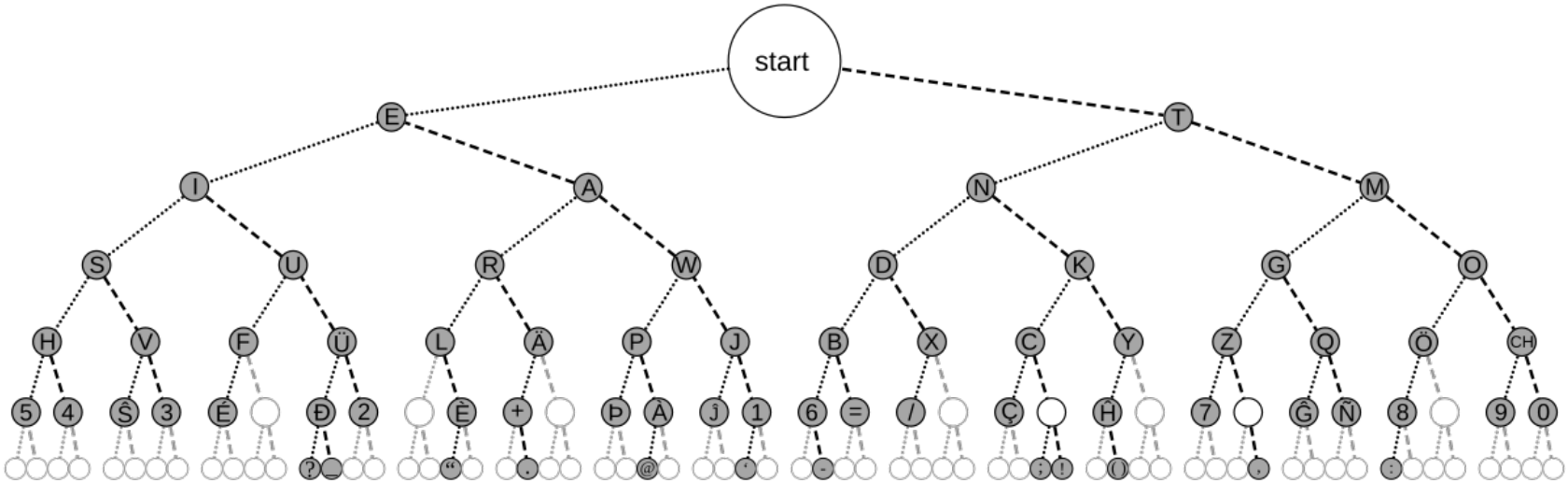


Kod Morse'a i drzewa binarne



Jak widać z obrazka, wszystkie znaki kodowane alfabetem Morse'a można zapisać jako węzły drzewa binarnego, w którym kod danego symbolu powstaje poprzez zapis drogi od korzenia do danego węzła: każdy wybór krawędzi w stronę lewego dziecka zapisujemy jako kropkę, a każdy wybór krawędzi w stronę prawego dziecka jako kreskę.

Kod Morse'a i drzewa binarne



Jak widać z obrazka, wszystkie znaki kodowane alfabetem Morse'a można zapisać jako węzły drzewa binarnego, w którym kod danego symbolu powstaje poprzez zapis drogi od korzenia do danego węzła: każdy wybór krawędzi w stronę lewego dziecka zapisujemy jako kropkę, a każdy wybór krawędzi w stronę prawego dziecka jako kreskę. Na przykład, od korzenia do litery G dochodzimy idąc najpierw dwa razy w prawo a potem raz w lewo, więc G w kodzie Morse'a zapisuje się jako $- - \cdot$.

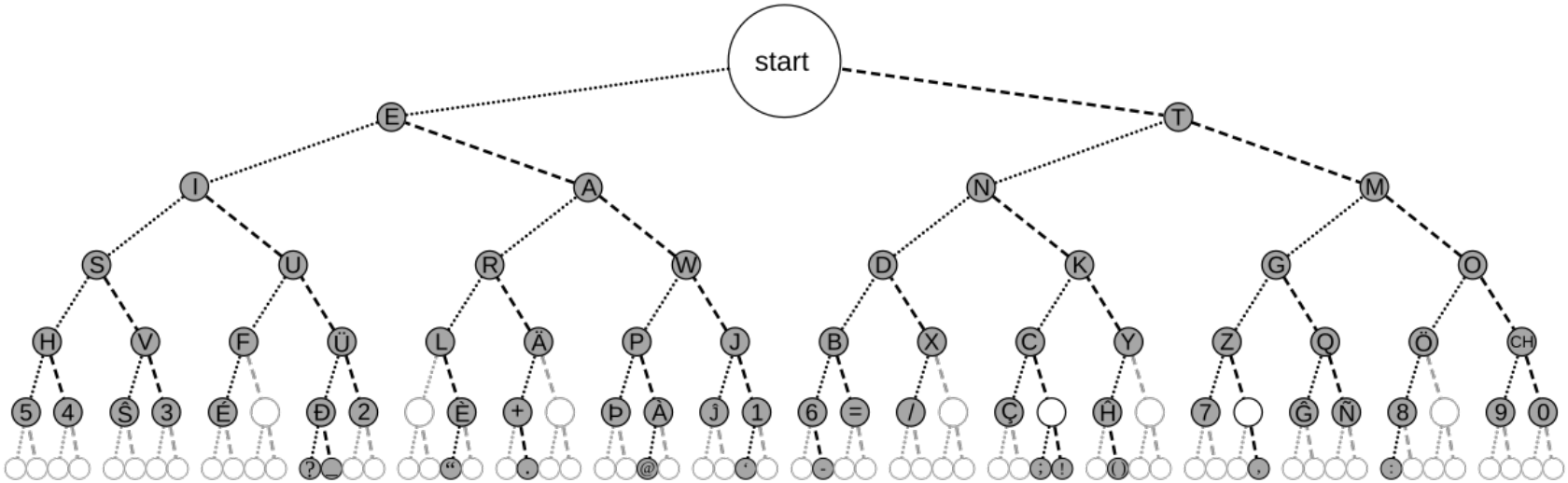
Kod Morse'a

Kod Morse'a poza szczególnymi wyjątkami związanymi z sytuacjami awaryjnymi (słynny sygnał SOS) obecnie ma znaczenie głównie historyczne, ale w XIX i XX wieku odegrał olbrzymią rolę w komunikacji telegraficznej i radiowej. Autor kodu słusznie założył, że często występującym w języku angielskim literom jak E ($\cdot$) lub T ($-$) należy przypisać krótsze ciągi niż rzadziej używanym jak F ($\cdot\cdot - \cdot$) albo Z ($- - \cdot \cdot$). Dzięki temu kod był w miarę „dobrze skompresowany" - średnia liczba znaków potrzebna do zakodowania danej litery była zmniejszona w stosunku do np. zupełnie losowego przypisania kodów znakom.

Kod Morse'a

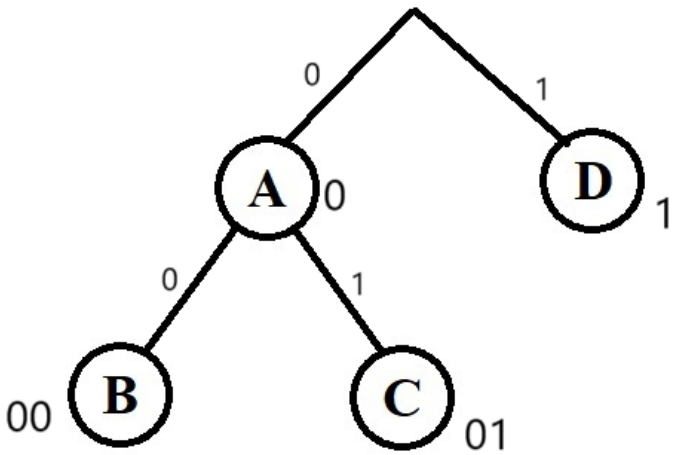
Kod Morse'a poza szczególnymi wyjątkami związanymi z sytuacjami awaryjnymi (słynny sygnał SOS) obecnie ma znaczenie głównie historyczne, ale w XIX i XX wieku odegrał olbrzymią rolę w komunikacji telegraficznej i radiowej. Autor kodu słusznie założył, że często występującym w języku angielskim literom jak E ($\cdot$) lub T ($-$) należy przypisać krótsze ciągi niż rzadziej używanym jak F ($\cdot\cdot - \cdot$) albo Z ($- - \cdot \cdot$). Dzięki temu kod był w miarę „dobrze skompresowany" - średnia liczba znaków potrzebna do zakodowania danej litery była zmniejszona w stosunku do np. zupełnie losowego przypisania kodów znakom.
Ale oczywiście, lepszą efektywność konwersji można uzyskać za pomocą teorii grafów...

Wady kodu Morse'a



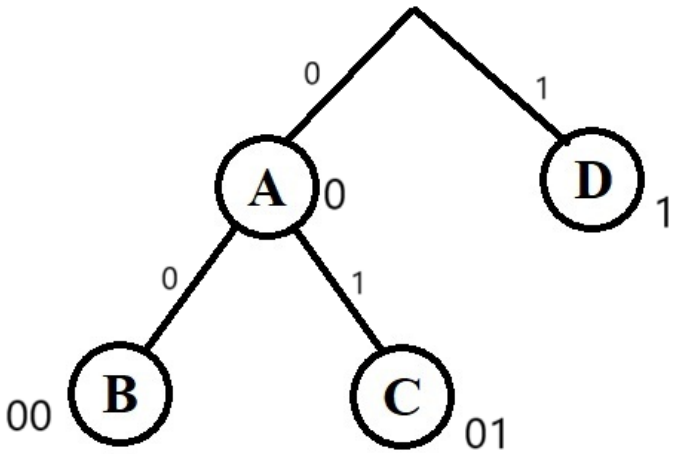
Podstawową wadą kodu Morse'a (poza nienajefektywniejszym rozmieszczeniem niektórych znaków) był brak jednoznaczności otrzymanego ciągu znaków. Na przykład – – – teoretycznie mogłoby oznaczać literę O, ale też np. ciągi liter MT, TM lub TTT. W alfabecie Morse'a rozwiązane jest to wprowadzeniem specjalnych przerw lub znaków pomiędzy transmisją liter, ale to de facto znacznie pogarsza efektywność konwersji (rozumianą jako liczbę symboli średnio wymaganych do przekazania wiadomości).

Błędne kodowanie



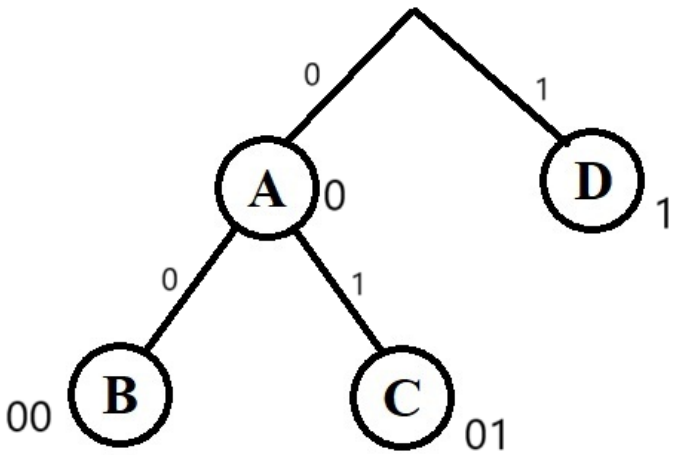
Wadę kodu Morse'a ilustruje w uproszczeniu (gdy kodujemy wiadomości oparte na alfabecie o 4 znakach: ABCD) powyższe drzewo. Odtąd będziemy zakładać, że kodując symbol za pomocą drogi z korzenia do symbolu „krok w lewo" dodaje do kodu 0, a krok w prawo dodaje do kodu 1. Dlatego otrzymujemy kody liter zapisane przy symbolizujących je węzłach.

Błędne kodowanie



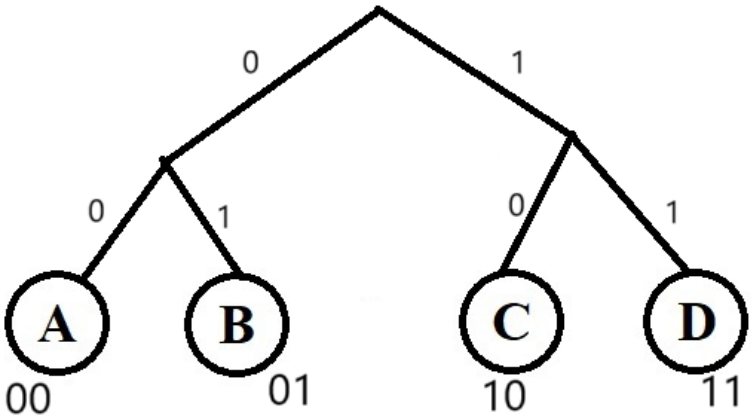
Jednak taka konstrukcja kodu powoduje niejednoznaczność zapisu: na przykład wiadomość 010001 może być odczytana jako ADAAAD, ale też jako CABD albo ADBC. Powodem niejednoznaczności jest to, że kod litery A jest początkiem kodu liter B i C.

Błędne kodowanie



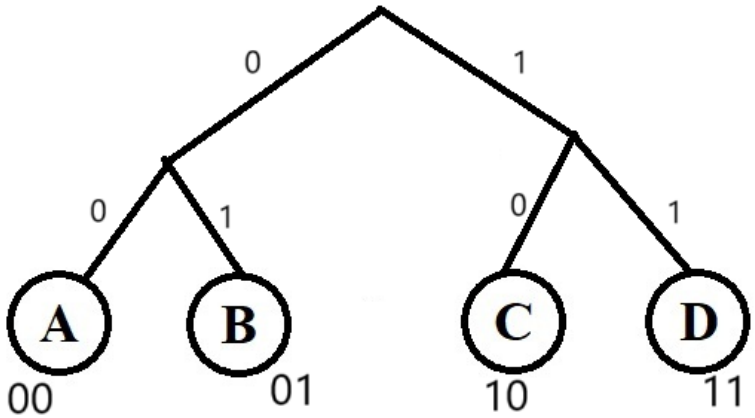
Jednak taka konstrukcja kodu powoduje niejednoznaczność zapisu: na przykład wiadomość 010001 może być odczytana jako ADAAAD, ale też jako CABD albo ADBC. Powodem niejednoznaczności jest to, że kod litery A jest początkiem kodu liter B i C. Dlatego, by kod miał sens, musi zachodzić warunek, że żaden kod nie może być początkiem innego. Przykładowo, 10 i 1011 nie mogą być jednocześnie kodami symboli.

Poprawne kodowanie



Okazuje się, że dodanie takiego warunku wystarcza, by kod stał się jednoznaczny. W tłumaczeniu na język drzew oznacza to, że żaden węzeł kodujący jakiś symbol nie stoi na drodze od korzenia do innego symbolu, co najprościej osiągnąć, gdy wszystkie takie węzły są liśćmi drzewa.

Poprawne kodowanie



Okazuje się, że dodanie takiego warunku wystarcza, by kod stał się jednoznaczny. W tłumaczeniu na język drzew oznacza to, że żaden węzeł kodujący jakiś symbol nie stoi na drodze od korzenia do innego symbolu, co najprościej osiągnąć, gdy wszystkie takie węzły są liśćmi drzewa. Na przykład powyższe drzewo odczytuje problematyczny poprzednio kod 010001 jednoznacznie jako BAB.

Binarne kody prefiksowe

Dlatego będą nas interesować tak zwane binarne kody prefiksowe.

Binarny kod prefiksowy

Binarnym kodem prefiksowym nazywamy kod oparty na drzewie binarnym z wyróżnionym korzeniem w którym:
I. Symbole języka są zakodowane jako liście.
II. Każdy wierzchołek nie będący liściem na dokładnie dwoje dzieci.

Binarne kody prefiksowe

Dlatego będą nas interesować tak zwane binarne kody prefiksowe.

Binarny kod prefiksowy

Binarnym kodem prefiksowym nazywamy kod oparty na drzewie binarnym z wyróżnionym korzeniem w którym:
I. Symbole języka są zakodowane jako liście.
II. Każdy wierzchołek nie będący liściem na dokładnie dwoje dzieci.

Drugie założenie tej definicji wynika z czystej efektywności: gdyby nie było spełnione, mielibyśmy do czynienia z dodawaniem dodatkowych krawędzi, a więc zbędnych dodatkowych zer i jedynek do kodu.

Binarne kody prefiksowe

Dlatego będą nas interesować tak zwane binarne kody prefiksowe.

Binarny kod prefiksowy

Binarnym kodem prefiksowym nazywamy kod oparty na drzewie binarnym z wyróżnionym korzeniem w którym:
I. Symbole języka są zakodowane jako liście.
II. Każdy wierzchołek nie będący liściem na dokładnie dwoje dzieci.

Drugie założenie tej definicji wynika z czystej efektywności: gdyby nie było spełnione, mielibyśmy do czynienia z dodawaniem dodatkowych krawędzi, a więc zbędnych dodatkowych zer i jedynek do kodu.
Jednak kodów binarnych jest wiele - jak wybrać z nich najoptymalniejszy i co ta optymalność miałaby oznaczać?

Zasada optymalizacji kodów prefiksowych

Podstawowa metoda optymalizacji binarnych kodów prefiksowych oparta jest na intuicji Morse'a: chcemy generalnie zapisywać kody typowych wiadomości w jak najmniejszej liczbie bitów (jedynek i zer). Innymi słowy, naszym celem jest znalezienie wśród kodów prefiksowych takiego, który minimalizuje długość średniego „słowa kodowego", czyli wartości oczekiwanej liczby bitów potrzebnych do zakodowania jednego znaku kodowanego alfabetu.

Zasada optymalizacji kodów prefiksowych

Podstawowa metoda optymalizacji binarnych kodów prefiksowych oparta jest na intuicji Morse'a: chcemy generalnie zapisywać kody typowych wiadomości w jak najmniejszej liczbie bitów (jedynek i zer). Innymi słowy, naszym celem jest znalezienie wśród kodów prefiksowych takiego, który minimalizuje długość średniego „słowa kodowego", czyli wartości oczekiwanej liczby bitów potrzebnych do zakodowania jednego znaku kodowanego alfabetu.
Osiągnąć to możemy przez odpowiednie przypisywanie znaków wierzchołkom - te, które występują częściej w wysyłanej wiadomości (lub wiadomościach) powinny być generalnie kodowane mniejszą liczbą bitów niż te, które występują rzadko.

Optymalizacja kodu i drzewa

Formalizując opis z poprzedniego slajdu, załóżmy, że dodatkowo dla każdego znaku alfabetu *i* znamy prawdopodobieństwo jego wystąpienia w kodowanym tekście (np. częstotliwość występowania litery w słowach danego języka albo po prostu - obliczamy to dla konkretnej wiadomości).

Optymalizacja kodu i drzewa

Formalizując opis z poprzedniego slajdu, załóżmy, że dodatkowo dla każdego znaku alfabetu i znamy prawdopodobieństwo jego wystąpienia w kodowanym tekście (np. częstotliwość występowania litery w słowach danego języka albo po prostu - obliczamy to dla konkretnej wiadomości). Prawdopodobieństwo to będzie wagą w_i węzła odpowiadającego temu znakowi w drzewie z wagami.

Optymalizacja kodu i drzewa

Formalizując opis z poprzedniego slajdu, załóżmy, że dodatkowo dla każdego znaku alfabetu i znamy prawdopodobieństwo jego wystąpienia w kodowanym tekście (np. częstotliwość występowania litery w słowach danego języka albo po prostu - obliczamy to dla konkretnej wiadomości). Prawdopodobieństwo to będzie wagą w_i węzła odpowiadającego temu znakowi w drzewie z wagami. Zauważmy, że poziom tego węzła l_i jest równy liczbie bitów, które są potrzebne do zakodowania tego znaku.

Optymalizacja kodu i drzewa

Formalizując opis z poprzedniego slajdu, załóżmy, że dodatkowo dla każdego znaku alfabetu i znamy prawdopodobieństwo jego wystąpienia w kodowanym tekście (np. częstotliwość występowania litery w słowach danego języka albo po prostu - obliczamy to dla konkretnej wiadomości). Prawdopodobieństwo to będzie wagą w_i węzła odpowiadającego temu znakowi w drzewie z wagami.
Zauważmy, że poziom tego węzła l_i jest równy liczbie bitów, które są potrzebne do zakodowania tego znaku.
Zatem wartość oczekiwana liczby bitów potrzebnych do zakodowania jednego znaku alfabetu jest równa wadze naszego drzewa:

$$W(T) = \sum_{i=1}^{n} l_i w_i.$$

Optymalizacja kodu i drzewa

Zatem wartość oczekiwana liczby bitów potrzebnych do zakodowania jednego znaku alfabetu jest równa wadze naszego drzewa:

$$W(T) = \sum_{i=1}^{n} l_i w_i.$$

Optymalizacja kodu i drzewa

Zatem wartość oczekiwana liczby bitów potrzebnych do zakodowania jednego znaku alfabetu jest równa wadze naszego drzewa:

$$W(T) = \sum_{i=1}^{n} l_i w_i.$$

Zatem wyznaczenie optymalnego kodu prefiksowego sprowadza się do wyznaczenia drzewa binarnego o minimalnej wadze i o liściach z wagami $w_1, \ldots, w_n$, gdzie w_i są odpowiednimi prawdopodobieństwami wystąpienia symboli w alfabecie. Taki kod nazywa się kodem Huffmana i jest wyznaczany za pomocą algorytmu Huffmana.

Zagadnienie Huffmana: przykład

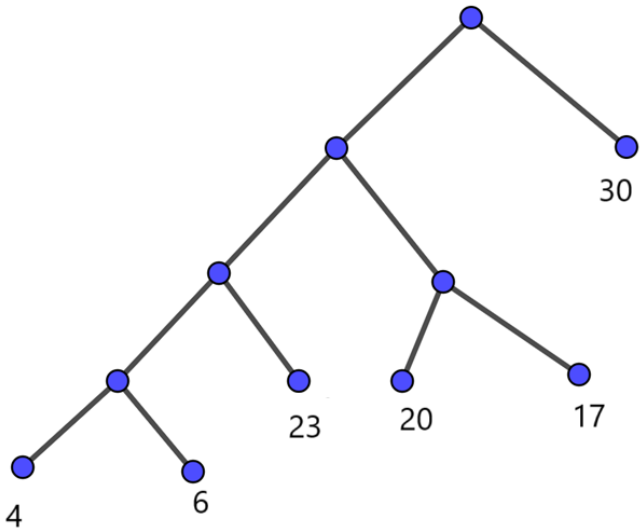
Zadanie

Wyznaczyć kod Huffmana dla alfabetu złożonego z liter A,B,C,D,E,F o częstotliwościach występowania odpowiednio: $(30\%, 4\%, 20\%, 6\%, 23\%, 17\%)$.

Innymi słowy, chcemy wyznaczyć drzewo binarne o minimalnej wadze, gdy jego liście mają wagi odpowiednio $0,3$; $0,04$; $0,2$; $0,06$; $0,23$ i $0,17$. Dla ułatwienia zapisu, przemnożę wszystkie wagi przez 100, żeby zamiast tego mieć wagi (30, 4, 20, 6, 23, 17). Nie wpłynie to na rozwiązanie zadania, gdyż waga W w wyjściowym problemie jest minimalna wtedy i tylko wtedy gdy waga $100W$ jest minimalna po przemnożeniu wag przez 100.

Propozycje rozwiązań

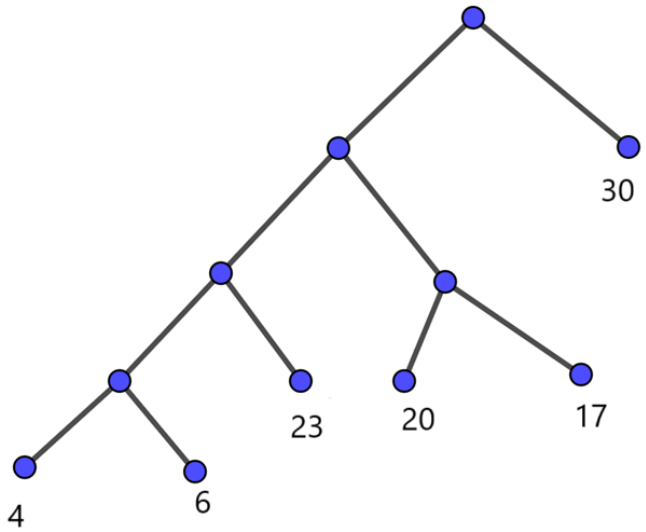
Przykładowe drzewa binarnych kodów prefiksowych naszego zadania:

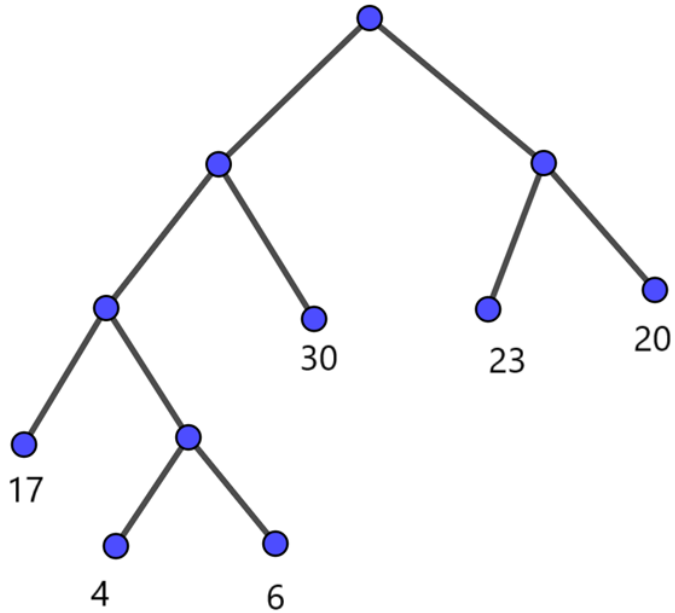


$$W(T) = 1 \cdot 30 + 3 \cdot (23 + 20 + 17) +$$
$$+ 4 \cdot (4 + 6) = 250.$$

Propozycje rozwiązań

Przykładowe drzewa binarnych kodów prefiksowych naszego zadania:



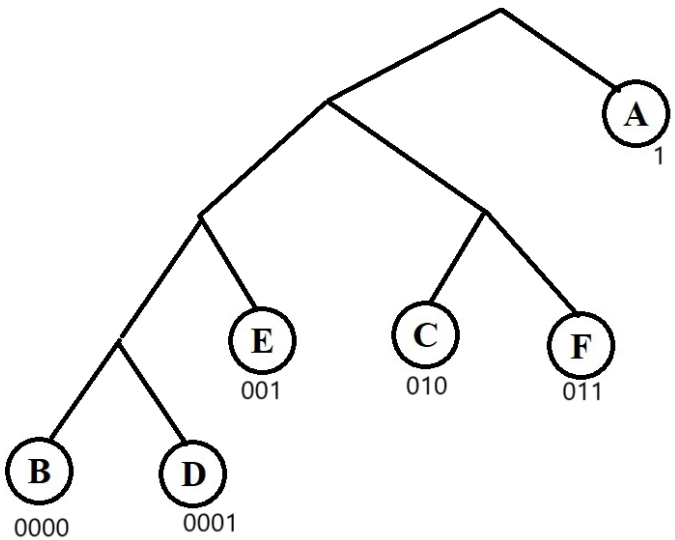


$$W(T) = 1{\cdot}30{+}3{\cdot}(23{+}20{+}17){+}$$
$$+4 \cdot (4+6) = 250.$$

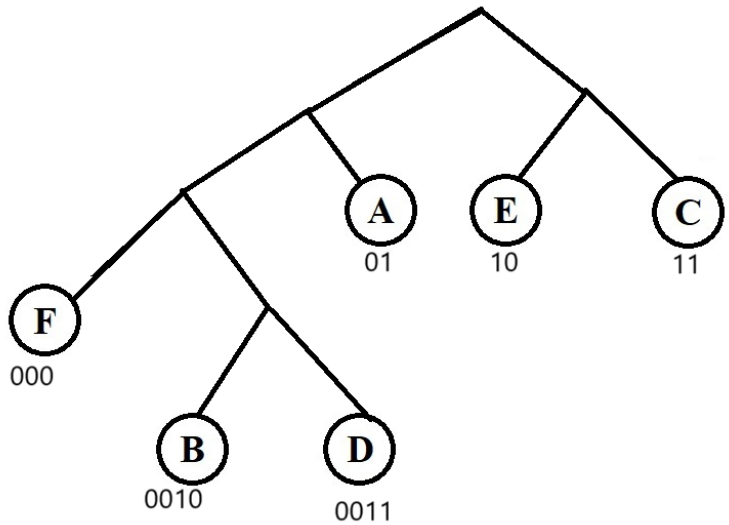
$$W(T) = 2{\cdot}(30{+}23{+}20){+}3{\cdot}17{+}$$
$$+4 \cdot (4+6) = 237.$$

Propozycje rozwiązań

Drzewo po lewej ma oczywiście za dużą wagę (2,5) jako średnią liczbę bitów na znak, by być optymalne. Drzewo po prawej jest lepsze, bo tworzy kod o średniej liczbie 2,37 bitów kodujących znak. Ale czy nie ma lepszego?



$W(T) = 250.$



$W(T) = 237.$

Algorytm Huffmana

HUFFMAN(w)

Dane: $n \geqslant 2$, ciąg $w = (w_1, w_2, \ldots, w_n)$ liczb nieujemnych.
Zmienne: $T(w)$ - drzewo, w' - ciąg.

Algorytm Huffmana

HUFFMAN(w)

Dane: $n \geqslant 2$, ciąg $w = (w_1, w_2, \ldots, w_n)$ liczb nieujemnych.
Zmienne: $T(w)$ - drzewo, w' - ciąg.

- I. Jeśli $n = 2$, to $T(w)$ - drzewo o złożone z korzenia i dwóch liści o wagach w_1 i w_2.

Algorytm Huffmana

HUFFMAN(w)

Dane: $n \geqslant 2$, ciąg $w = (w_1, w_2, \ldots, w_n)$ liczb nieujemnych.
Zmienne: $T(w)$ - drzewo, w' - ciąg.

- I. Jeśli $n = 2$, to $T(w)$ - drzewo o złożone z korzenia i dwóch liści o wagach w_1 i w_2.
- II. Jeśli $n > 2$, znajdujemy najmniejsze 2 wyrazy ciągu w np. u i v i definiujemy ciąg w' jako ciąg w, w którym wyrazy u i v zastępujemy przez jeden wyraz $u + v$.

Algorytm Huffmana

HUFFMAN(w)

Dane: $n \geqslant 2$, ciąg $w = (w_1, w_2, \ldots, w_n)$ liczb nieujemnych.
Zmienne: $T(w)$ - drzewo, w' - ciąg.

- I. Jeśli $n = 2$, to $T(w)$ - drzewo o złożone z korzenia i dwóch liści o wagach w_1 i w_2.
- II. Jeśli $n > 2$, znajdujemy najmniejsze 2 wyrazy ciągu w np. u i v i definiujemy ciąg w' jako ciąg w, w którym wyrazy u i v zastępujemy przez jeden wyraz $u + v$.
- IIa. Wykonujemy HUFFMAN(w'), otrzymując drzewo $T(w')$.

Algorytm Huffmana

HUFFMAN(w)

Dane: $n \geqslant 2$, ciąg $w = (w_1, w_2, \ldots, w_n)$ liczb nieujemnych.
Zmienne: $T(w)$ - drzewo, w' - ciąg.

- I. Jeśli $n = 2$, to $T(w)$ - drzewo o złożone z korzenia i dwóch liści o wagach w_1 i w_2.
- II. Jeśli $n > 2$, znajdujemy najmniejsze 2 wyrazy ciągu w np. u i v i definiujemy ciąg w' jako ciąg w, w którym wyrazy u i v zastępujemy przez jeden wyraz $u + v$.
- IIa. Wykonujemy HUFFMAN(w'), otrzymując drzewo $T(w')$.
- IIIa. Tworzymy $T(w)$, zastępując w $T(w')$ liść wagi $u + v$ poddrzewem złożonym z korzenia i dwóch liści o wagach u i v.

Algorytm Huffmana

HUFFMAN(w)

Dane: $n \geqslant 2$, ciąg $w = (w_1, w_2, \ldots, w_n)$ liczb nieujemnych.
Zmienne: $T(w)$ - drzewo, w' - ciąg.

- I. Jeśli $n = 2$, to $T(w)$ - drzewo o złożone z korzenia i dwóch liści o wagach w_1 i w_2.
- II. Jeśli $n > 2$, znajdujemy najmniejsze 2 wyrazy ciągu w np. u i v i definiujemy ciąg w' jako ciąg w, w którym wyrazy u i v zastępujemy przez jeden wyraz $u + v$.
- IIa. Wykonujemy HUFFMAN(w'), otrzymując drzewo $T(w')$.
- IIIa. Tworzymy $T(w)$, zastępując w $T(w')$ liść wagi $u + v$ poddrzewem złożonym z korzenia i dwóch liści o wagach u i v.
- **Rezultat**: $T(w)$ - drzewo binarne o minimalnej wadze dla ciągu wag liści $w = (w_1, w_2, \ldots, w_n)$.

Algorytm Huffmana - interpretacja

Algorytm Huffmana działa rekurencyjnie: dodajemy do siebie najmniejsze pary elementów ciągu wag liści, tworząc z dwóch elementów jeden i zapisując, z jakim ciągiem mamy do czynienia po każdym kroku (i najlepiej, jak powstał ostatni element).

Algorytm Huffmana - interpretacja

Algorytm Huffmana działa rekurencyjnie: dodajemy do siebie najmniejsze pary elementów ciągu wag liści, tworząc z dwóch elementów jeden i zapisując, z jakim ciągiem mamy do czynienia po każdym kroku (i najlepiej, jak powstał ostatni element). W ten sposób skracamy ciąg wag o 1 w każdym kroku, aż dojdziemy do sytuacji, gdy zostają w nim dwa elementy.

Algorytm Huffmana - interpretacja

Algorytm Huffmana działa rekurencyjnie: dodajemy do siebie najmniejsze pary elementów ciągu wag liści, tworząc z dwóch elementów jeden i zapisując, z jakim ciągiem mamy do czynienia po każdym kroku (i najlepiej, jak powstał ostatni element). W ten sposób skracamy ciąg wag o 1 w każdym kroku, aż dojdziemy do sytuacji, gdy zostają w nim dwa elementy. Wtedy tworzymy z nich drzewo o korzeniu i dwóch liściach.

Algorytm Huffmana - interpretacja

Algorytm Huffmana działa rekurencyjnie: dodajemy do siebie najmniejsze pary elementów ciągu wag liści, tworząc z dwóch elementów jeden i zapisując, z jakim ciągiem mamy do czynienia po każdym kroku (i najlepiej, jak powstał ostatni element). W ten sposób skracamy ciąg wag o 1 w każdym kroku, aż dojdziemy do sytuacji, gdy zostają w nim dwa elementy. Wtedy tworzymy z nich drzewo o korzeniu i dwóch liściach. Następnie rozbudowujemy to drzewo, wracając do coraz dłuższych ciągów i „rozwijając" kolejne liście w poddrzewa, odwracając proces sumowania węzłów w jeden.

Algorytm Huffmana - przykład

Zadanie

Wyznaczyć kod Huffmana dla alfabetu złożonego z liter A,B,C,D,E,F o częstotliwościach występowania odpowiednio: $(30\%, 4\%, 20\%, 6\%, 23\%, 17\%)$.

Zgodnie w wcześniejszym ustaleniem, wykonujemy HUFFMAN$(30, 4, 20, 6, 23, 17)$.

Algorytm Huffmana - przykład

Zadanie

Wyznaczyć kod Huffmana dla alfabetu złożonego z liter A,B,C,D,E,F o częstotliwościach występowania odpowiednio: $(30\%, 4\%, 20\%, 6\%, 23\%, 17\%)$.

Zgodnie w wcześniejszym ustaleniem, wykonujemy HUFFMAN$(30, 4, 20, 6, 23, 17)$. $n = 6 > 2$ więc „scalamy" najmniejsze elementy ciągu wag którymi są

Algorytm Huffmana - przykład

Zadanie

Wyznaczyć kod Huffmana dla alfabetu złożonego z liter A,B,C,D,E,F o częstotliwościach występowania odpowiednio: $(30\%, 4\%, 20\%, 6\%, 23\%, 17\%)$.

Zgodnie w wcześniejszym ustaleniem, wykonujemy HUFFMAN$(30, 4, 20, 6, 23, 17)$. $n = 6 > 2$ więc „scalamy" najmniejsze elementy ciągu wag którymi są 4 i 6, przechodząc do wykonania: HUFFMAN$(30, 20, 23, 17, 4 + 6 = 10)$.

Algorytm Huffmana - przykład

Zadanie

Wyznaczyć kod Huffmana dla alfabetu złożonego z liter A,B,C,D,E,F o częstotliwościach występowania odpowiednio: $(30\%, 4\%, 20\%, 6\%, 23\%, 17\%)$.

Zgodnie w wcześniejszym ustaleniem, wykonujemy HUFFMAN$(30, 4, 20, 6, 23, 17)$. $n = 6 > 2$ więc „scalamy" najmniejsze elementy ciągu wag którymi są 4 i 6, przechodząc do wykonania: HUFFMAN$(30, 20, 23, 17, 4 + 6 = 10)$. $n = 5 > 2$ więc „scalamy" najmniejsze elementy ciągu wag którymi są

Algorytm Huffmana - przykład

Zadanie

Wyznaczyć kod Huffmana dla alfabetu złożonego z liter A,B,C,D,E,F o częstotliwościach występowania odpowiednio: $(30\%, 4\%, 20\%, 6\%, 23\%, 17\%)$.

Zgodnie w wcześniejszym ustaleniem, wykonujemy HUFFMAN$(30, 4, 20, 6, 23, 17)$. $n = 6 > 2$ więc „scalamy" najmniejsze elementy ciągu wag którymi są 4 i 6, przechodząc do wykonania: HUFFMAN$(30, 20, 23, 17, 4 + 6 = 10)$. $n = 5 > 2$ więc „scalamy" najmniejsze elementy ciągu wag którymi są 10 i 17, przechodząc do wykonania: HUFFMAN$(30, 20, 23, 10 + 17 = 27)$.

Algorytm Huffmana - przykład

Zadanie

Wyznaczyć kod Huffmana dla alfabetu złożonego z liter A,B,C,D,E,F o częstotliwościach występowania odpowiednio: $(30\%, 4\%, 20\%, 6\%, 23\%, 17\%)$.

Zgodnie w wcześniejszym ustaleniem, wykonujemy HUFFMAN$(30, 4, 20, 6, 23, 17)$. $n = 6 > 2$ więc „scalamy" najmniejsze elementy ciągu wag którymi są 4 i 6, przechodząc do wykonania: HUFFMAN$(30, 20, 23, 17, 4 + 6 = 10)$. $n = 5 > 2$ więc „scalamy" najmniejsze elementy ciągu wag którymi są 10 i 17, przechodząc do wykonania: HUFFMAN$(30, 20, 23, 10 + 17 = 27)$. $n = 4 > 2$ więc „scalamy" najmniejsze elementy ciągu wag którymi są

Algorytm Huffmana - przykład

Zadanie

Wyznaczyć kod Huffmana dla alfabetu złożonego z liter A,B,C,D,E,F o częstotliwościach występowania odpowiednio: $(30\%, 4\%, 20\%, 6\%, 23\%, 17\%)$.

Zgodnie w wcześniejszym ustaleniem, wykonujemy HUFFMAN$(30, 4, 20, 6, 23, 17)$. $n = 6 > 2$ więc „scalamy" najmniejsze elementy ciągu wag którymi są 4 i 6, przechodząc do wykonania: HUFFMAN$(30, 20, 23, 17, 4 + 6 = 10)$. $n = 5 > 2$ więc „scalamy" najmniejsze elementy ciągu wag którymi są 10 i 17, przechodząc do wykonania: HUFFMAN$(30, 20, 23, 10 + 17 = 27)$. $n = 4 > 2$ więc „scalamy" najmniejsze elementy ciągu wag którymi są 20 i 23, przechodząc do wykonania: HUFFMAN$(30, 27, 20 + 23 = 43)$.

Algorytm Huffmana - przykład

(...) przechodząc do wykonania: HUFFMAN$(30, 27, 20 + 23 = 43)$.

Algorytm Huffmana - przykład

(...) przechodząc do wykonania: HUFFMAN$(30, 27, 20 + 23 = 43)$. $n = 3 > 2$ więc „scalamy” najmniejsze elementy ciągu wag którymi są

Algorytm Huffmana - przykład

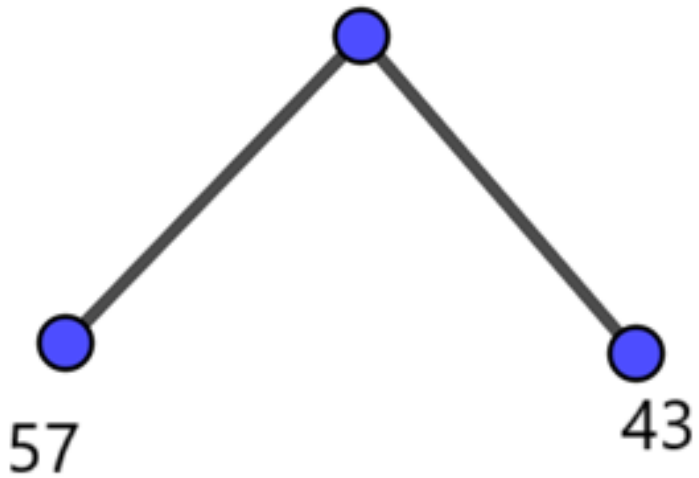
(...) przechodząc do wykonania: HUFFMAN$(30, 27, 20 + 23 = 43)$.
$n = 3 > 2$ więc „scalamy” najmniejsze elementy ciągu wag którymi są 30 i 27, przechodząc do wykonania: HUFFMAN$(30 + 27 = 57, 43)$.
Ponieważ $n = 2$, możemy zacząć konstruować optymalne drzewo.

Algorytm Huffmana - przykład

(...) przechodząc do wykonania: HUFFMAN$(30, 27, 20 + 23 = 43)$.
$n = 3 > 2$ więc „scalamy” najmniejsze elementy ciągu wag którymi są 30 i 27, przechodząc do wykonania: HUFFMAN$(30 + 27 = 57, 43)$.
Ponieważ $n = 2$, możemy zacząć konstruować optymalne drzewo.
Dla ułatwienia konstrukcji zapamiętujemy sobie kolejne ciągi wag i to, w jaki sposób powstawały: $(30, 4, 20, 6, 23, 17)$;
$(30, 20, 23, 17, 4 + 6 = 10)$; $(30, 20, 23, 10 + 17 = 27)$;
$(30, 27, 20 + 23 = 43)$; $(30 + 27 = 57, 43)$.

Algorytm Huffmana - przykład

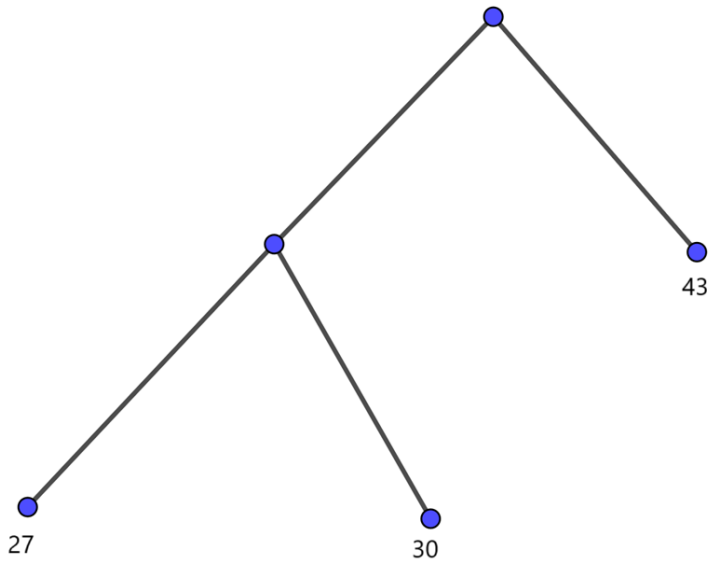
$(30, 4, 20, 6, 23, 17)$; $(30, 20, 23, 17, 4 + 6 = 10)$;
$(30, 20, 23, 10 + 17 = 27)$; $(30, 27, 20 + 23 = 43)$; $(30 + 27 = 57, 43)$.
HUFFMAN(57,43) tworzy drzewo:



Ponieważ wartość 57 powstała jako suma wag 30 i 27, możemy uzyskać rozwiązanie HUFFMAN$(30, 27, 43)$ rozbijając węzeł z wagą 57 w poddrzewo o 2 liściach z wagami 30 i 27.

Algorytm Huffmana - przykład

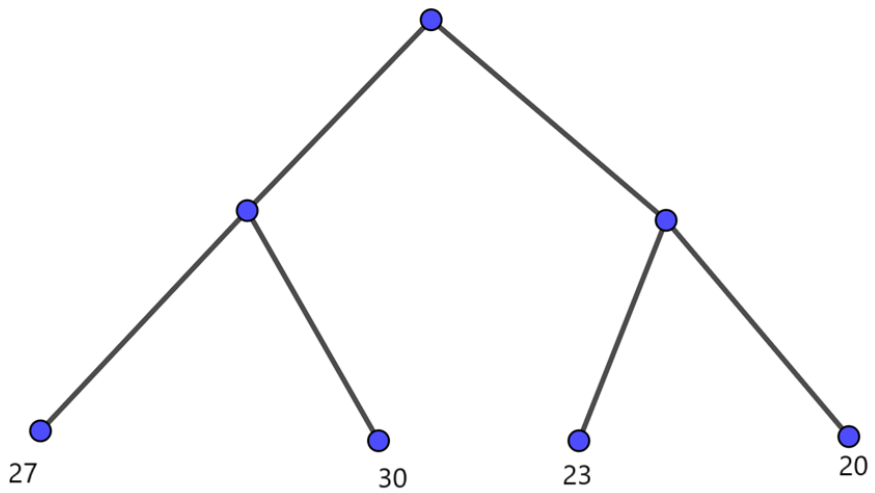
$(30, 4, 20, 6, 23, 17)$; $(30, 20, 23, 17, 4 + 6 = 10)$;
$(30, 20, 23, 10 + 17 = 27)$; $(30, 27, 20 + 23 = 43)$; $(30 + 27 = 57, 43)$.
HUFFMAN$(30, 27, 43)$ tworzy drzewo:



Ponieważ wartość 43 powstała jako suma wag 20 i 23, możemy uzyskać rozwiązanie HUFFMAN$(30, 20, 23, 27)$ rozbijając węzeł z wagą 43 w poddrzewo o 2 liściach z wagami 20 i 23.

Algorytm Huffmana - przykład

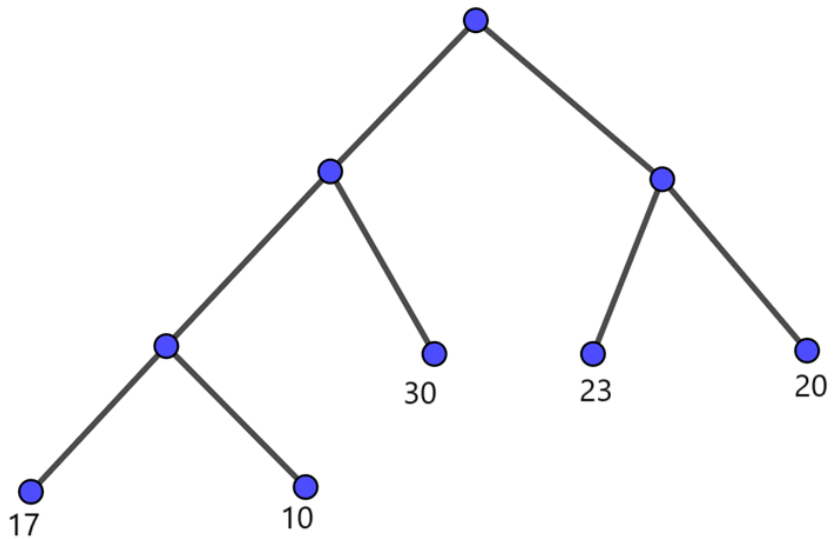
$(30, 4, 20, 6, 23, 17)$; $(30, 20, 23, 17, 4 + 6 = 10)$;
$(30, 20, 23, 10 + 17 = 27)$; $(30, 27, 20 + 23 = 43)$; $(30 + 27 = 57, 43)$.
HUFFMAN$(30, 20, 23, 27)$ tworzy drzewo:



Ponieważ wartość 27 powstała jako suma wag 10 i 17, możemy uzyskać rozwiązanie HUFFMAN$(30, 20, 23, 17, 10)$ rozbijając węzeł z wagą 27 w poddrzewo o 2 liściach z wagami 10 i 17.

Algorytm Huffmana - przykład

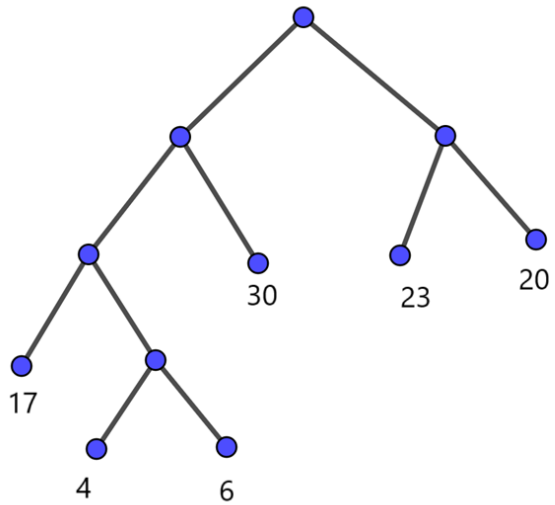
$(30, 4, 20, 6, 23, 17)$; $(30, 20, 23, 17, 4 + 6 = 10)$;
$(30, 20, 23, 10 + 17 = 27)$; $(30, 27, 20 + 23 = 43)$; $(30 + 27 = 57, 43)$.
HUFFMAN$(30, 20, 23, 17, 10)$ tworzy drzewo:



I wreszcie ponieważ wartość 10 powstała jako suma wag 4 i 6 , możemy uzyskać rozwiązanie HUFFMAN$(30, 4, 20, 6, 23, 17)$ rozbijając węzeł z wagą 10 w poddrzewo o 2 liściach z wagami 4 i 6.

Algorytm Huffmana - przykład

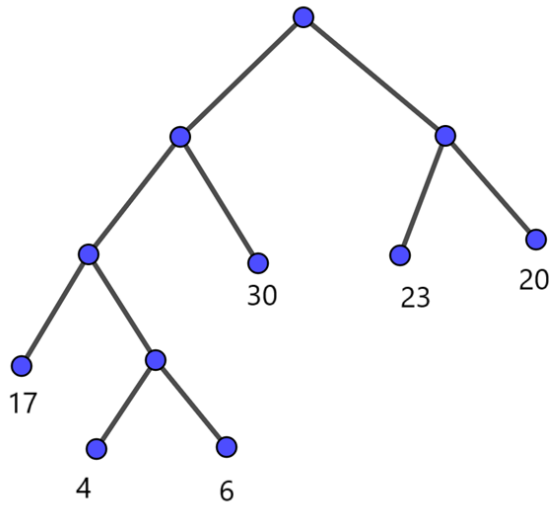
$(30, 4, 20, 6, 23, 17)$; $(30, 20, 23, 17, 4 + 6 = 10)$;
$(30, 20, 23, 10 + 17 = 27)$; $(30, 27, 20 + 23 = 43)$; $(30 + 27 = 57, 43)$.
HUFFMAN$(30, 4, 20, 6, 23, 17)$ tworzy drzewo:



i kończy się wykonywać, gdyż to były dane wejściowe.

Algorytm Huffmana - przykład

$(30, 4, 20, 6, 23, 17)$; $(30, 20, 23, 17, 4 + 6 = 10)$;
$(30, 20, 23, 10 + 17 = 27)$; $(30, 27, 20 + 23 = 43)$; $(30 + 27 = 57, 43)$.
HUFFMAN$(30, 4, 20, 6, 23, 17)$ tworzy drzewo:



i kończy się wykonywać, gdyż to były dane wejściowe. Waga tego drzewa wynosi

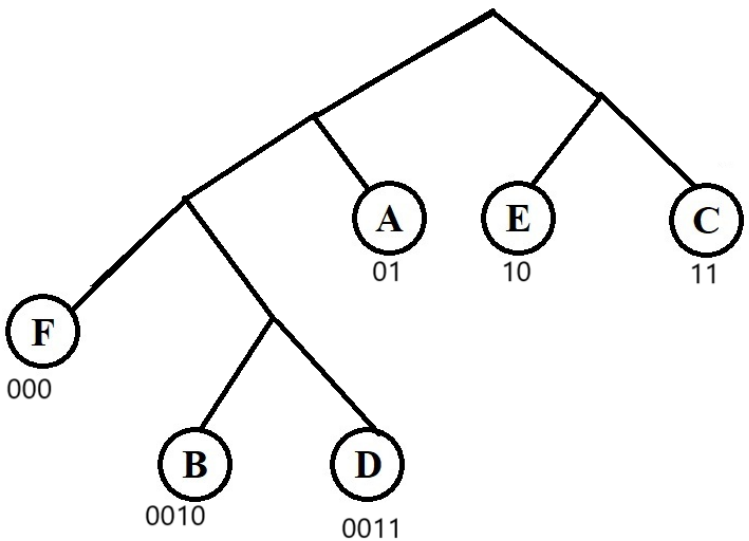
$$W(T) = 2 \cdot (30 + 23 + 20) + 3 \cdot 17 + 4 \cdot (4 + 6) = 237.$$

Algorytm Huffmana - przykład

Zadanie

Wyznaczyć kod Huffmana dla alfabetu złożonego z liter A,B,C,D,E,F o częstotliwościach występowania odpowiednio: $(30\%, 4\%, 20\%, 6\%, 23\%, 17\%)$.

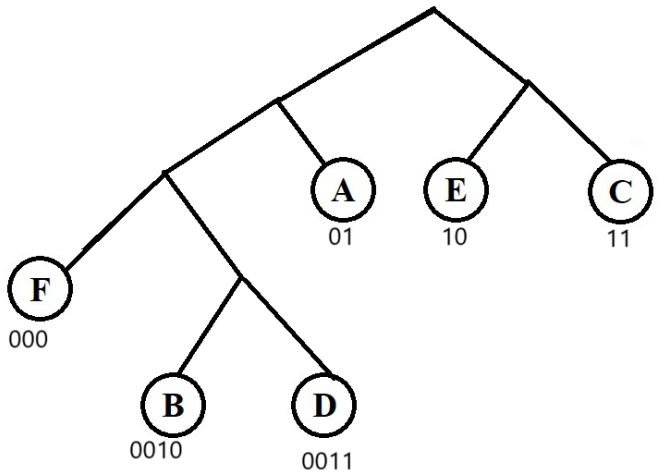
Podstawiając znaki alfabetu w miejsce ich wag/prawdopodobieństw otrzymamy drzewo kodowania:



Algorytm Huffmana - przykład

Zadanie

Wyznaczyć kod Huffmana dla alfabetu złożonego z liter A,B,C,D,E,F o częstotliwościach występowania odpowiednio: $(30\%, 4\%, 20\%, 6\%, 23\%, 17\%)$.



Na przykład, słowo CAFE będzie jednoznacznie zakodowane jako 110100010.

Algorytm Huffmana - przykład

Zadanie

Wyznaczyć kod Huffmana dla alfabetu złożonego z liter A,B,C,D,E,F o częstotliwościach występowania odpowiednio:
$(30\%, 4\%, 20\%, 6\%, 23\%, 17\%)$.

Oczywiście, jeśli jako wagi weźmiemy prawdziwe częstotliwości, to algorytm zadziała tak samo, ale waga drzewa, które wyjdzie, wyniesie 100 razy mniej, czyli 2,37.
Zatem stosując kodowanie z poprzedniego slajdu, otrzymamy kod, który średnio wymaga 2,37 bitów by zakodować jeden znak i wiedzę, że nie da się tego zrobić lepiej.

Algorytm Huffmana - uwagi

- Jeśli algorytm Huffmana będzie zadaniem na sprawdzianie, będę wymagać przedstawienia kroków pośrednich tzn. pośrednich ciągów wag i kolejno uzyskiwanych drzew, tak jak to zrobiłem na poprzednich slajdach.

Algorytm Huffmana - uwagi

- Jeśli algorytm Huffmana będzie zadaniem na sprawdzianie, będę wymagać przedstawienia kroków pośrednich tzn. pośrednich ciągów wag i kolejno uzyskiwanych drzew, tak jak to zrobiłem na poprzednich slajdach.
- Algorytm Huffmana nie jest do końca detetministyczny: jeśli w ciągu w' wystąpią kiedykolwiek dwie takie same wagi, możemy wybrać węzeł przypisany do dowolnej z nich. Ponadto, algorytm nie wskazuje, który z liści mamy umieszczać jako lewe, a który jako prawe poddrzewo. Dlatego poprawnie zastosowany algorytm Huffmana może wyznaczyć wiele różnych drzew - natomiast wszystkie z nich mają tę samą, minimalną wagę.

Algorytm Huffmana - uwagi

- Kod Huffmana zawsze jest kodem prefiksowym, więc jednoznacznym.

Algorytm Huffmana - uwagi

- Kod Huffmana zawsze jest kodem prefiksowym, więc jednoznacznym.
- Złożoność obliczeniowa algorytmu Huffmana to $O(n \log n)$, a kodowanie i dekodowanie znaków za jego pomocą - $O(n)$.

Algorytm Huffmana - uwagi

- Kod Huffmana zawsze jest kodem prefiksowym, więc jednoznacznym.
- Złożoność obliczeniowa algorytmu Huffmana to $O(n \log n)$, a kodowanie i dekodowanie znaków za jego pomocą - $O(n)$.
- Efektywniejszy kod Hoffmana można otrzymać analizując pary znaków alfabetu zamiast pojedynczych znaków.

Algorytm Huffmana - uwagi

- Kod Huffmana zawsze jest kodem prefiksowym, więc jednoznacznym.
- Złożoność obliczeniowa algorytmu Huffmana to $O(n \log n)$, a kodowanie i dekodowanie znaków za jego pomocą - $O(n)$.
- Efektywniejszy kod Hoffmana można otrzymać analizując pary znaków alfabetu zamiast pojedynczych znaków.
- Ciekawym wariantem problemu jest *dynamiczne* zagadnienie Huffmana, w którym prawdopodobieństwa poznajemy w miarę napływu danych i dynamicznie zmieniamy optymalny kod (np. algorytm Vittera).