**Algorytm Huffmana**

Algorytm kodowania Huffmana jest algorytmem bezstratnej kompresji danych. Działanie algorytmu Huffmana polega na określeniu prawdopodobieństw wystąpień P = {p1, ..., pn} symboli S = {x1, ..., xn} danego alfabetu.

Kodowanie Huffmana:

1. Określamy prawdopodobieństwo, bądź częstość wystąpienia dla każdego symbolu ze zbioru S.
2. Tworzymy listę drzew binarnych, przechowujących w węzłach symbol oraz prawdopodobieństwo (częstość) wystąpienia danego symbolu. Początkowo drzewa w liście będą stanowiły same korzenie.
3. Dopóki na liście jest więcej niż jedno drzewo, powtarzamy:
   1. Usuwamy z listy dwa drzewa z najmniejszymi prawdopodobieństwami (częstościami) wystąpień1
   2. Tworzymy nowe drzewo, przechowujące w korzeniu sumę prawdopodobieństw (częstości) wystąpień symboli drzew usuniętych w punkcie 3a. Usunięte w punkcie 3a drzewa stają się lewym i prawym poddrzewem nowo-utworzonego drzewa. Dodajemy nowe drzewo do listy.

Po zakończeniu powyższej procedury powstanie tzw. **drzewo Huffmana,** w którym w korzeniu przechowywane jest prawdopodobieństwo równe 1, a symbole do zakodowania wraz z ich prawdopodobieństwami (częstościami) wystąpień zapisane są w liściach.

Powstałe drzewo Huffmana posłuży do wyznaczenia kodów Huffmana, w następujący sposób:

1. Każdej lewej krawędzi każdego węzła przypisujemy wartość **0**, a każdej prawej krawędzi wartość **1**
2. Przechodzimy drzewo od gorzenia do każdego liścia, do kodu z punktu 1. dopisujemy **0**, gdy przechodzimy do lewego poddrzewa, natomiast **1** gdy przechodzimy do prawego poddrzewa.

Powstałe kody Huffmana stanowią kod dla każdego symbolu przechowywanego w liściu.

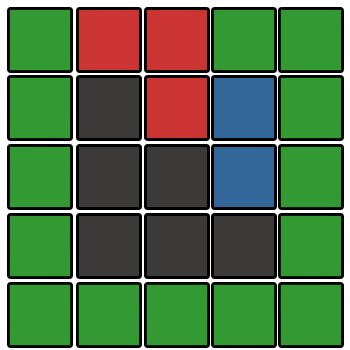
*Przykład*

Obrazek wielkość: 5\*5 = 25

Zbiór symboli: wartość koloru każdego piksela

Prawdopodobieństwo: ilość wystąpień każdego koloru w całym obrazie

Niech: **G** - zielony, **K** - czarny, **R** - czerwony, **B** - niebieski



1. Zliczamy ilość wystąpień każdego koloru

G - 14, K - 6, R - 3, B - 2

2) Tworzymy listę drzew.



3) Budujemy drzewo i wyznaczamy kody

|  |  |
| --- | --- |
| Bez nazwy-3 | Bez nazwy-5 |
| Bez nazwy-6 | Bez nazwy-6 |

Jak wynika z przykładu symbole o największej częstości występowania otrzymują kody najkrótsze, natomiast te które rzadko występują otrzymują kody dłuższe.

Dekodowanie pliku graficznego wygląda podobnie do kodowania.

Z wczytanego nagłówka obrazka zawierającego pary kolor - ilość wystąpień jesteśmy w stanie odbudować drzewo użyte do kodowania.

Po odbudowaniu drzewa algorytm dekodowania przebiega następująco

1. Z pliku wczytywana jest bajt zawierający ciągu wartości 0 i 1.
2. Sprawdzamy czy w drzewie istnieje symbol o kodzie Huffmana równym pierwszemu bitowi wczytanego bajtu w kroku 1.  
   Jeżeli symbol o kodzie nie znaleziony -> krok 3.  
   Jeżeli symbol o kodzie znaleziony -> krok 4.
3. Pobieramy kolejny bit (wartość 0/1) z wczytanego w kroku 1 bajtu, dołączamy do istniejącego ciągu wartości szukanego kodu i ponownie przeszukujemy drzewo (krok 2).
4. Jeżeli w drzewie został znaleziony symbol o kodzie szukanym to go pobieramy i zerujemy ciągu naszego klucza przeszukiwań drzewa.
5. Jeżeli cały plik wczytany to kończymy algorytm.

*Działanie algorytmu Huffmana w projekcie*

Implementacja algorytmu Huffmana w projekcie przebiega według powyższej procedury. Przy czym dla obrazka o jednolitym kolorze dodawany jest do drzewa pusty węzeł - kod koloru otrzymuje wówczas wartość 1. Zapis do pliku polega na zastąpieniu danego koloru odpowiednim dla niego kodem. Kody wpisywane do pliku nie są rozszerzanego do bajtów - wpisywane są ‘bitami’. Podejście takie wydłuża czas kodowanie oraz znacznie bardziej czas dekodowania, ale pozwala uzyskać wiele wyższy stopień kompresji - dlatego zdecydowano się na realizację takiego podejścia.

Kodowanie wartości poszczególnych pikseli bardzo dobrze sprawdza się dla plików graficznych zawierających wiele pikseli tego samego koloru, jednakże dla obrazów mających zróżnicowane kolory algorytm Huffmana może dać słabszy stopień kompresji - w zależności od poziomu zróżnicowania kolorów.

W celu zdekodowania obrazka wymagane jest odtworzenie drzewa - identycznego do użytego podczas kodowania - co jest realizowane poprzez zapisanie w nagłówku obrazka kolorów oraz odpowiadającym im liczby wystąpień (co niestety powoduje rozrost wynikowego rozmiaru pliku graficznego).

**Algorytm 1: Upakowanie bitów - 8 bitów na 4 bity**

Cel: Stratna kompresja danych

Działanie algorytmu:

Kompresja:

1) Pobranie składowej koloru z pixela.

2) Obcięcie jej wartości do 4 najbardziej znaczących bitów.

3) Umieszczenie jej w odpowiedniej części 8bitowego kontenera

* Jeżeli kontener jest pusty, dane wstawiane są do początkowej jego części.
* Jeżeli kontener zawiera już dane, następne dane są wstawiane do drugiej jego połówki.

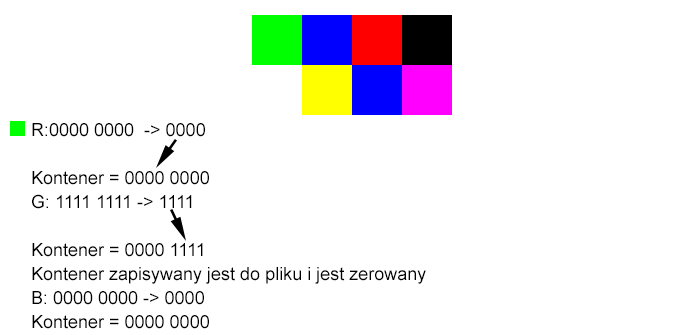
1. Sprawdzany jest stan kontenera, jeżeli jest zapełniony, kontener zapisywany jest do pliku wynikowego.
2. Wybierana jest kolejna składowa, lub kolejny pixel i następuje przejście do kroku 2.

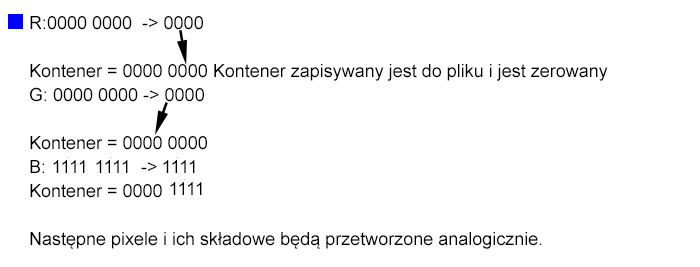
Dekompresja:

1. Pobierany jest 8bitowy kontener z bufora
2. Uruchamiana jest pętla (przechodząca po wszystkich pixelach) a w niej pętla przechodząca po 3 składowych( czerwona, zielona, niebieska)
3. Następuje pobranie 8bitowego kontenera z buffora
4. Sprawdzanie stanu bufora
5. Jeżeli z kontenera nie było pobranych żadnych danych, pobierane są z niego pierwsze 4 bity
6. Jeżeli pobrano już jakieś dane z kontenera, pobierane są pozostałe( ostatnie) 4 bity
7. Wstawianie danych wybranej ( w pętli) składowej pixela
8. Jeżeli pobrano wszystkie dane z kontenera, następuje pobranie kolejnego kontenera

Zalety: Mała złożoność obliczeniowa, mała ilość zajmowanej pamięci

Wady: Strata jakości

Kompresja



Dekompresja

