Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Matematyczny
specjalność: informatyczna

Łukasz Koszyk

Wybrane algorytmy kompresji obrazów.

Praca magisterska
napisana pod kierunkiem
prof. Waldemara Hebischa

Wrocław 2014

Oświadczam, że pracę magisterską wykonałem samodzielnie
i zgłaszam ją do oceny.

Data:..... Podpis autora pracy:.....

Oświadczam, że praca jest gotowa do oceny przez recenzenta.

Data:..... Podpis opiekuna pracy:.....

Spis treści

Wstęp

Jest rok 2014. Cyfrowa rewolucja trwa. Cyfryzują się biblioteki, urzędy, archiwa państwowe, telewizja naziemna. Ten trend nie ominął też zwykłych ludzi. Zdjęć nie robi się już aparatami na kliszę. Stare Kodaki i Zenity spoczęły na strychu, ustępując miejsca aparatom cyfrowym, głównie kompaktowym, a te z kolei właśnie wypierane są przez aparaty wbudowane w telefony i smartfony. Pudła po butach wypełnione zdjęciami zastępują dziś karty pamięci, nośniki optyczne i dyski twarde. Pieczałowicie zdobione albumy rodzinne zastępuje się albumami elektronicznymi przechowywanymi w chmurze. Zdjęcia z wakacji, którymi niegdyś byliśmy torturowani przez znajomych podczas odwiedzin, teraz raczej umieszcza się w internecie, na portalu społecznościowym. Zdjęcia bliskich trzymamy dziś w telefonie, aniżeli w portfelu, który w epoce transakcji elektronicznych stracił na znaczeniu.

Cywilizacja zachodnia wytworzyła w swojej masie nową kulturę - kulturę obrazu. Obraz jest na każdym kroku, i to nie tylko w postaci reklam, którymi obklejone są miasta. Ludzie między sobą zaczęli się dzielić nie tyle przemyśleniami, co komunikować obrazami, które przy obecnej dostępności technologii bardzo łatwo stworzyć i przesłać. Myślimy słowami - kluczami, ripostami, obrazami. Zamiast tekstów: statystyki w postaci infografik, diagramów i grafów, życzenia zdawkowe opatrzone obrazem, zamiast opisów - obraz przedstawiający stany emocjonalne. Nawet dowcipy skraca się do zdjęcia i riposty pod spodem. Wszystko w postaci atrakcyjnej, szybkiej do odczytania. I każdy, kto ma choćby telefon z dostępem do internetu może zrobić zdjęcie: tego gdzie jest, co je, czym jedzie, lub stworzyć jakąś grafikę, a następnie włączyć aplikację Facebook czy Instagram, i w mgnieniu oka opublikować to w internecie. Z drugiej strony mamy zjawisko memów internetowych, najczęściej obrazków opatrzonych krótką ripostą, powstających natychmiast po wydarzeniu generującym kontrowersje i zainteresowanie: koncert, wybory, ulewa, matury itp. Ludzie dzielą się nimi między sobą za pośrednictwem portali społecznościowych, a najciekawsze publikowane są przez prasę i telewizję.

Wobec takiej powszechności informacji, jaką jest obraz, warto zastanowić się nad tym, dlaczego najbardziej powszechnym jest format jpeg? Jak to możliwe, że te obrazy zajmują tak mało miejsca na dysku? Jakim procesom poddawane są zdjęcia czy memy kompresowane tym algorytmem? Co aparat cyfrowy zapisuje w pliku jpg? Dlaczego mając kadr zapisany w pliku jpg i raw, ten pierwszy jest kilkudziesięciokrotnie mniejszy od drugiego? Dlaczego niektóre obrazki w internecie, przy wolnym łączu, widoczne są od razu w niskiej jakości, która stopniowo się poprawia?

Poniekąd odpowiedzią na powyższe pytania jest ta praca. Porusza ona problematykę kompresji obrazów, omawia różne techniki, ze szczególnym ukierunkowaniem na standard JPEG. Składa się z czterech głównych części.

Pierwsza to omówienie podstawowych pojęć oraz mechanizmu powstawania zdjęć w aparacie fotograficznym. Jest to punkt wyjścia do omówienia motywacji wprowadzania kompresji obrazów do urządzeń, a następnie omówione zostają dwie główne metody: stratna i bezstratna, oraz podstawowe idee przyświecające każdej z nich.

W drugiej części omówione są techniki bezstratne poprzez kodowanie entropijne: kodami Huffmana w wersji statycznej i dynamicznej, oraz kodami arytmetycznymi również w wersjach statycznej i dynamicznej, wraz z przykładami. Są to podstawowe techniki obecne w wielu kodekach używanych współcześnie.

Trzecią część rozpoczyna wprowadzenie do podstaw technik kodowania stratnego. Omówione są fizjologiczne aspekty postrzegania obrazu przez człowieka, i związane z tym wnioski, stanowiące podstawy pracy kodeków stratnych transformatowych. Następnie wprowadzona zostaje transformata Fouriera: wyprowadzony zostaje wzór na dyskretną transformatę Fouriera, a potem omówione są jej najważniejsze cechy, zilustrowane przykładami. Kolejnym etapem jest uzasadnienie wprowadzenia transformaty kosinusowej, wyprowadzenie wersji ciągłej, oraz wyprowadzenie wzoru na dyskretną transformatę kosinusową. Podobnie, jak w przypadku transformaty Fouriera - podsumo-

wane to zostanie omówieniem podstawowych właściwości na przykładach. Trzecią część kończy przedstawienie ogólnej idei stratnego kodowania transformatowego. Omówione zostają poszczególne kroki na potrzeby dalszej części pracy, ale pomijając szczegóły dotyczące ściśle transformaty kosinusowej, opis będzie pasował również do kompresji z użyciem innych technik.

Ostatnia część, to omówienie algorytmu JPEG: każdego z trybów pracy, ze szczególnym uwzględnieniem trybu sekwencyjnego, jako domyślnego. Jest to swoiste podsumowanie całej pracy, gdyż JPEG wykorzystuje wszystkie omawiane techniki kompresji.

Pracę kończy omówienie syntaktyki standardu JPEG oraz informacje i referencje dotyczące innych metod kompresji obrazów.

1 Podstawy teorii informacji

W niniejszej pracy będziemy się zajmować metodami kompresji obrazów. Aby to zrobić, konieczne jest wprowadzenie kilku pojęć.

1.1 Sygnał wielowymiarowy

Sygnałem wielowymiarowym (w tym przypadku n-wymiarowym) nazywać będziemy sygnał opisany funkcją wielu zmiennych niezależnych [1, s. 58]

$$g(x_1, x_2, ..., x_n)$$

Przykładem sygnału dwuwymiarowego jest obraz monochromatyczny. Weźmy funkcję $f(x, y)$ będącą naszym sygnałem. Niech x to będzie współrzędna pozioma obrazu, a y – pionowa. Wartość funkcji f w punkcie (x, y) to będzie ilość światła emitowanego z tego punktu – czyli wartość skalarna.

Innym przykładem sygnału dwuwymiarowego jest obraz kolorowy, z tym że wartością funkcji f w punkcie (x, y) nie będzie skalar jak w obrazie monochromatycznym, tylko wektor składający się ze skalarów: dla obrazu zapisanego w przestrzeni barw RGB będzie to wektor mający trzy wartości natężenia światła: koloru czerwonego, koloru zielonego oraz koloru niebieskiego (dla przestrzeni $CMYK$ – cztery barwy).

Przykładem sygnału trójwymiarowego będzie funkcja trzech zmiennych $f(x, y, t)$, przy czym dwie pierwsze zmienne są jak wyżej, natomiast zmienna t to czas. Zatem wartość w punkcie będzie zależna również od czasu. Przykładem jest np. materiał wideo (monochromatyczny lub kolorowy – w zależności od przyjętej definicji).

1.2 Sygnał dyskretny, a sygnał ciągły

Rozróżnia się sygnały dyskretne i sygnały ciągłe. Sygnał ciągły jest to funkcja niezależnych zmiennych, z których każda może przyjąć dowolną wartość rzeczywistą z pewnego przedziału. Zaś sygnałem dyskretnym nazywać będziemy funkcję, której argumenty da się ponumerować liczbami całkowitymi.

Jest jeszcze trzeci typ sygnału – mieszany. W tym przypadku pewne zmienne są zmiennymi dyskretnymi, a pewne zmiennymi o wartościach rzeczywistych z pewnego przedziału.

Przykładem sygnału mieszanego jest sygnał telewizji analogowej wyświetlanej na telewizorze kineskopowym monochromatycznym. Obraz składa się z kilkuset linii które są wyświetlane jedna po drugiej. W tym przypadku sygnał jest ciągły względem linii poziomych, a dyskretny względem kierunku pionowego i czasu [1, s. 61].

1.3 Próbkowanie sygnału i zakłócenia z tym związane

Załóżmy, że chcemy zrobić zdjęcie. Rzeczywisty obraz jest sygnałem ciągłym. Mamy do dyspozycji urządzenie mające skończoną liczbę punktów, którymi potrafimy zmierzyć wartość natężenia światła (lampa analizująca lub matryca światłoczuła). Próbkowaniem obrazu nazwiemy pomiar parametrów światła w każdym z punktów. Obraz utworzony przez próbkowanie będzie miał skończone wymiary i skończoną dokładność. Aby dobrze przybliżyć obraz, należy użyć odpowiedniej liczby próbek, o czym mówi poniższe twierdzenie:

1.3.1 Twierdzenie o próbkowaniu

Twierdzenie Kotelnikova-Nyquista-Shannona-Wittakera [9, s. 1]:

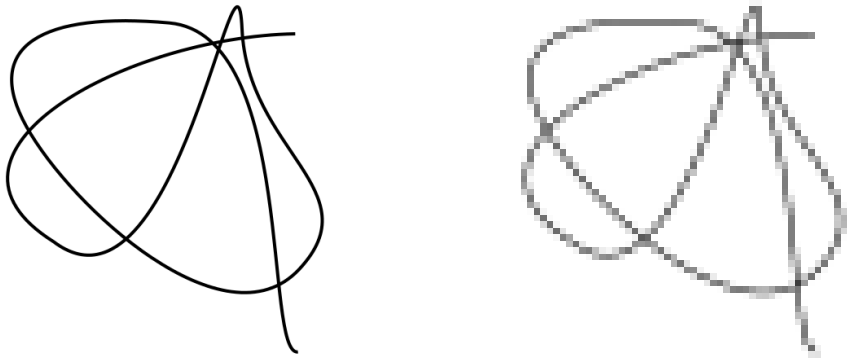
Aby wiernie odtworzyć z próbek sygnał ciągły, częstotliwość próbkowania musi być większa niż dwukrotność najwyższej składowej częstotliwości w próbkowanym sygnale.

1.3.2 Częstotliwość Nyquista

Jest to częstotliwość najwyższej składowej harmonicznej w sygnale, którą to składową chcemy odtworzyć bez zniekształceń w procesie przetwarzania sygnału z postaci dyskretnej, do postaci ciągłej.

1.3.3 Aliasing

Przypuśćmy, że nie możemy próbkować sygnału z częstotliwością co najmniej dwukrotnie większą od częstotliwości Nyquista. Wówczas próbki sygnału o wyższych częstotliwościach mogą być błędnie interpretowane jako próbki o niskiej częstotliwości, w konsekwencji czego w przetworzonym sygnale pojawią się składowe niskich częstotliwości, czyli tzw. aliasy. Nakładają się one na widmo pozostałego sygnału, przez co niemożliwym staje się wierne odtworzenie sygnału ciągłego z takiego zbioru próbek, zaś w samych obrazach pojawiają się artefakty:



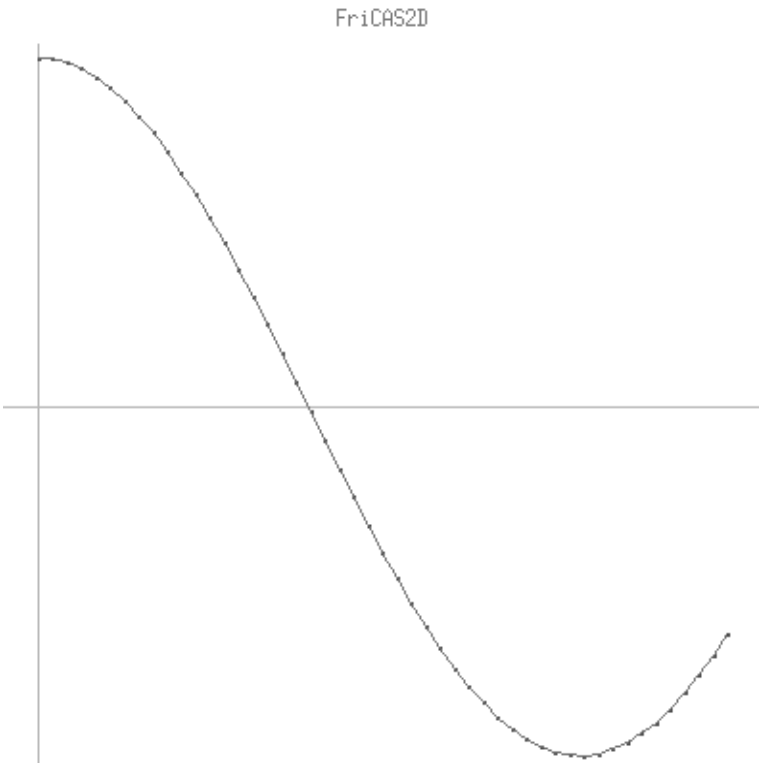
(a) Prawidłowa rozdzielczość (b) Za mała rozdzielczość

Rysunek 1: Przykład aliasingu

Załóżmy na chwilę, że nasz sygnał jest funkcją jednowymiarową. W celu unaocznienia jak ważne jest prawidłowe próbkowanie sygnału, posłużymy się prostym przykładem. W tym celu użyjemy systemu algebry komputerowej FriCAS.[1] Próbkujemy sygnał niewystarczającą liczbą próbek:

```
1 a:=2*%pi*48.0/4+1
2 draw((cos(a*1.00*x)), x=0..4)
```

W efekcie po odtworzeniu otrzymamy sygnał:



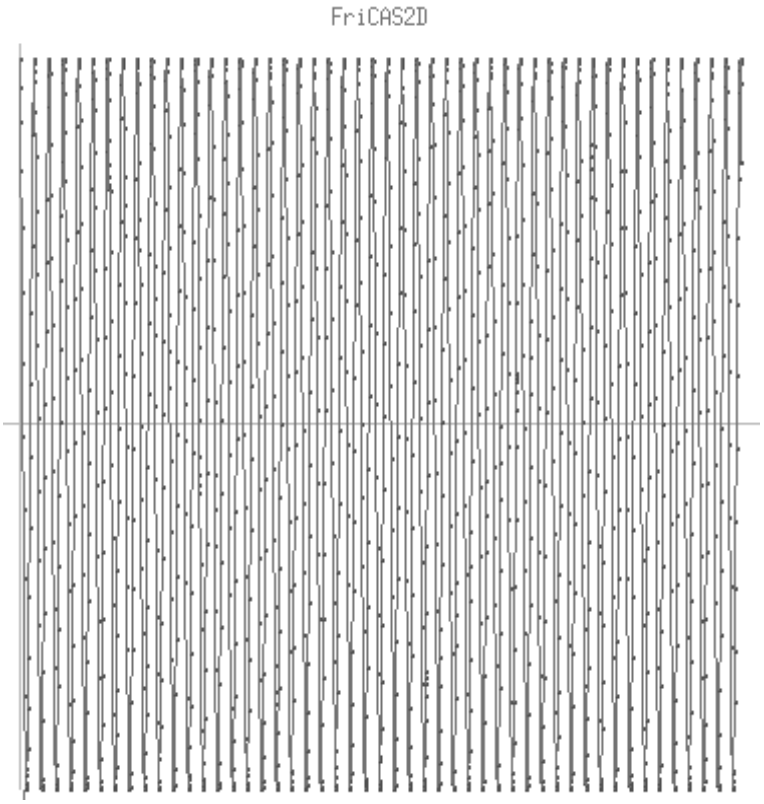
Rysunek 2: Wynik działania pierwszego skryptu.

[1] http://fricas.sourceforge.net/, rozwijany przez prof. Waldemara Hebischa

Spróbkujmy go minimalnie lepiej:

```
1 a:=2*%pi*48.0/4+1
2 draw((cos(a*1.01*x)), x=0..4)
```

Wówczas otrzymamy sygnał mocno odbiegający od poprzedniego:



Rysunek 3: Wynik działania drugiego skryptu.

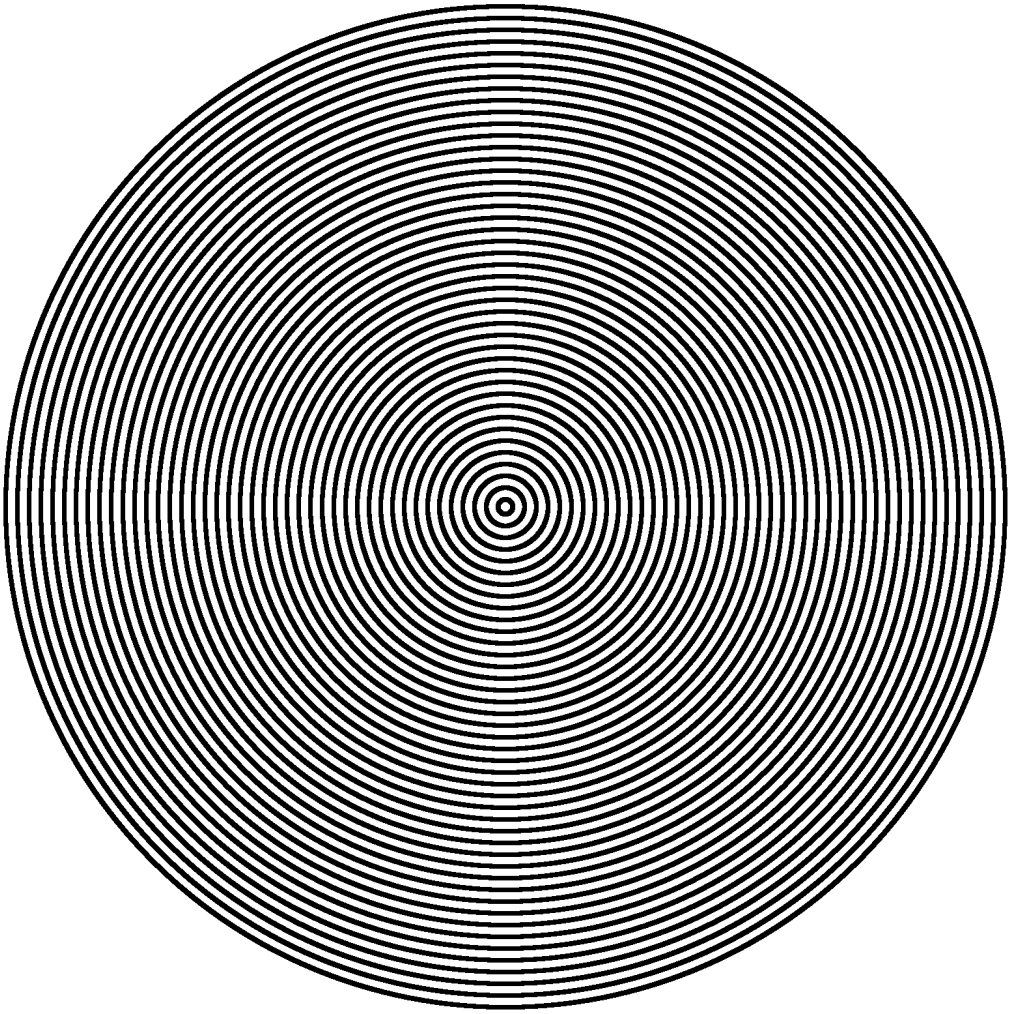
Na tym polega sedno aliasingu i istotność częstotliwości Nyquista: próbkując sygnał niedostateczną liczbą próbek, po odtworzeniu możemy otrzymać kompletnie inny sygnał.

Walczyć z aliasingiem można na różne sposoby. Oprócz oczywistego - zwiększenia częstotliwości próbkowania, można nałożyć filtr na widmo sygnału tak, aby pasmo filtru było ograniczone do częstotliwości Nyquista. W praktyce na ogół nie da się uniknąć aliasingu, i ten rodzaj zakłóceń będzie towarzyszył sygnałowi niemal zawsze.

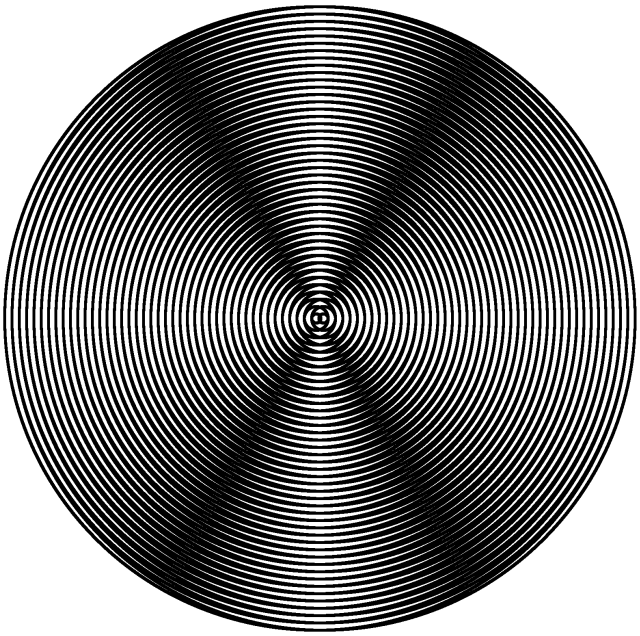
1.3.4 Mory

W zakresie, który nas interesuje, jest to niekorzystne zjawisko (ma zastosowanie w metrologii). Powstaje ono gdy nachodzą na siebie regularne punkty lub wzory. Wówczas powstaje zjawisko optyczne (np. podczas nałożenia się dwóch firan) lub w przypadku grafiki komputerowej - artefaktów, polegających na powstawaniu geometrycznych wzorów, prążków. Zjawiska tego często nie da się uniknąć, ale można je minimalizować. W przypadku mor na wyświetlaczach ciekłokrystalicznych i kineskopowych - należy je skalibrować. W fotografii i telewizji należy unikać obiektów (prawie)przezroczystych o geometrycznej fakturze i wzorzystej odzieży, a jeśli to niemożliwe - tak kadrować ujęcia, z jednoczesnym użyciem wyższej rozdzielczości, aby zminimalizować ryzyko. Powinno się również unikać robienia zdjęć pracującym telewizorom i monitorom. W grafice komputerowej zjawisko to powstaje przy źle dobranej rozdzielczości - gdy rozmiar piksela jest porównywalny z okresem wzorów. Przy zmniejszaniu fotografii przedstawiających budynki z cegły (lub mających inne równie regularne wzorki), gałęzie, źdźbła trawy itp. należy zachować ostrożność i umiejętnie dobrać parametry.

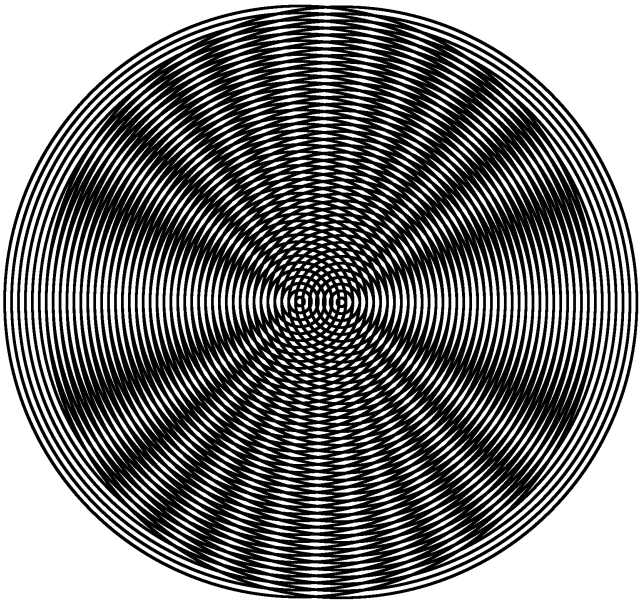
Prześledźmy teraz przykłady prążków mory (aby uniknąć problemów przy druku obrazki wstawione są w większych rozmiarach). Jako pierwsze - nałożenie na siebie okręgów: z użyciem programu GIMP na przezroczystej warstwie utworzono kilkadziesiąt okręgów równej grubości, w równej odległości. Następnie po zduplikowaniu warstwy, przesunięto jedną względem drugiej.



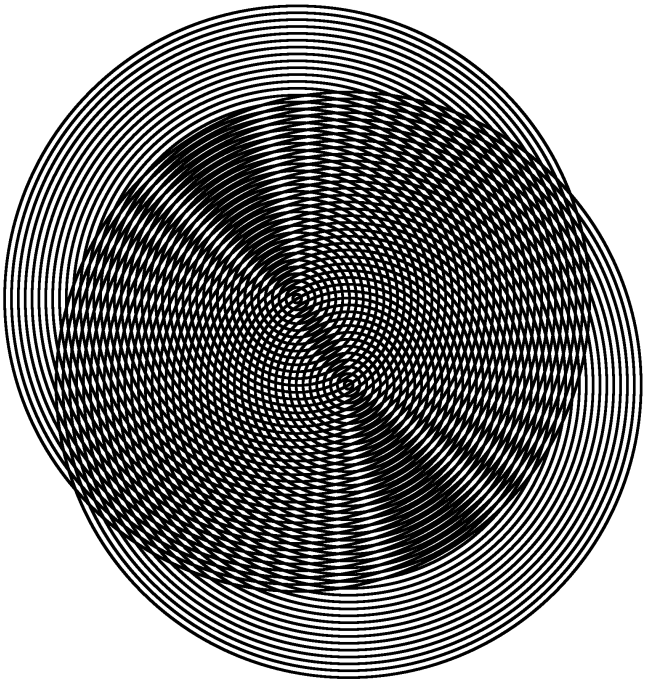
Rysunek 4: Pojedyncza warstwa



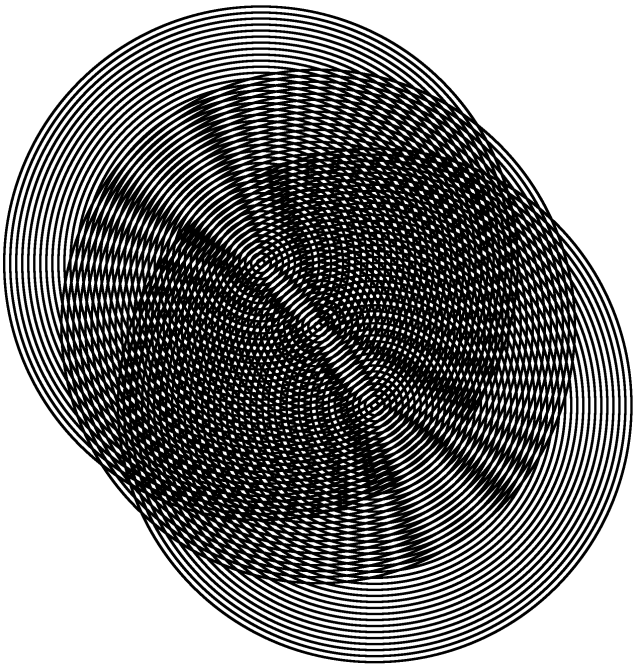
Rysunek 5: Niewielkie przesunięcie dwóch warstw względem siebie



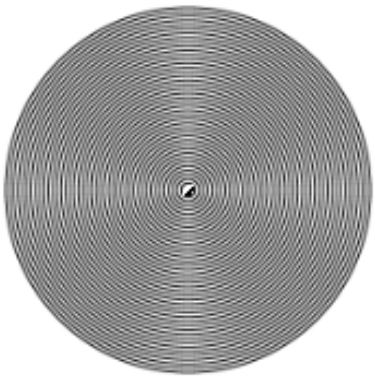
Rysunek 6: Większe przesunięcie dwóch warstw względem siebie



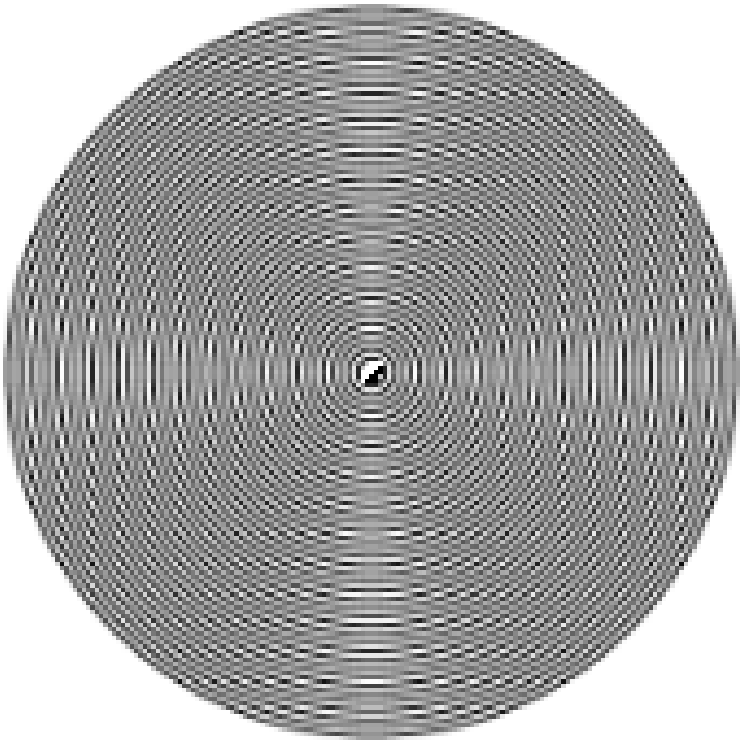
Rysunek 7: Dalsze przesunięcie dwóch warstw względem siebie



Rysunek 8: Przesunięcie trzech warstw względem siebie



Rysunek 9: Niewłaściwie dobrana rozdzielczość - pojedyncza warstwa.



Rysunek 10: Po powiększeniu. Widoczne prostokątne mory.

1.4 Sygnał cyfrowy

Sygnałem cyfrowym będziemy nazywać sygnał dyskretny i skwantyzowany.

1.5 Jak powstają zdjęcia w cyfrowym aparacie?

Mamy przed sobą krajobraz, i chcemy zrobić zdjęcie. Upraszczając sytuację: załączamy aparat[2], zdejmujemy dekiel zasłaniający soczewkę obiektywu i rozpoczynamy kadrowanie. Aparat sprawdził w tym czasie stan techniczny podzespołów, zasilania oraz czy ma zainstalowaną kartę pamięci, i zgłosił gotowość. Światło wpada do obiektywu, gdzie jest skupiane przez zespół soczewek, przechodzi przez przysłonę, kolejny zespół soczewek, a po wyjściu z obiektywu wpada do korpusu aparatu, gdzie odbija się od lustra, a następnie przez pryzmat pentagonalny kierowane jest do wizjera, gdzie po przejściu przez kolejne soczewki trafia do naszego oka.

Po ustawieniu kadru, parametrów ekspozycji wciskamy spust migawki do połowy (dla uproszczenia zakładamy że aparat pracuje w trybie półautomatycznym), urządzenie (w zależności od ustawień) dobiera czas naświetlania, ustawia przysłonę, wyostrza i blokuje się na tych parametrach zgłaszając gotowość. Wciśnięcie spustu do końca natychmiast uruchamia procedurę wykonania zdjęcia: zamykana jest migawka[3] i podnoszone jest lustro. Otwierana jest migawka i światło oświetla matrycę. Rozpoczyna się naświetlanie (które trwa od 1/16000 sekundy do 30-60 sekund lub „nieskończoności”[4]). Matryca składa się z pikseli pokrytych filtrami w taki sposób, że do piksela dociera światło o określonej długości w trzech kolorach: czerwonym, zielonym i niebieskim. Fotony wpadają w piksele matrycy powodując ładowanie „kondensatorków”[5] każdego piksela proporcjonalnie do natężenia światła. Po zakończeniu naświetlania migawka jest zamykana, lustro opuszczane, a ładunek

[2]Zakładamy, że jest to lustrzanka.

[3]Nie w każdym aparacie znajduje się migawka mechaniczna. W części tanich urządzeń stosuje się „migawkę” elektroniczną

[4]Tryb B (BULB) dostępny w droższych urządzeniach - naświetlanie trwa dopóki spust migawki jest wciśnięty.

[5]Dla uproszczenia przyjmujemy, że mamy do czynienia z mikroskopijnymi kondensatorkami. W rzeczywistości są to elektrody, w których wytworzono studnię potencjału.

w „kondensatorkach" stanowi teraz informację. Po wzmocnieniu, zmierzeniu i konwersji dostajemy cyfrowy sygnał. Mamy zatem informację o natężeniu światła w każdym z punktów w jednym z trzech kolorów: czerwonym, zielonym, niebieskim. Sygnał ten jest zapisywany odpowiednio w pamięci operacyjnej, następnie procesor (w zależności od ustawień) albo zapisuje surowe dane w pliku .raw[6] albo przetwarza uzyskane dane, czyli: koryguje kolory i ustawia balans bieli, nakłada filtry wybrane przez użytkownika, zmniejsza oraz kompresuje zgodnie z ustawieniami, a następnie zapisuje na karcie pamięci[7]. W międzyczasie otwierana jest migawka. Po zakończeniu zapisu pliku, aparat zgłasza gotowość do następnego zdjęcia.

W aparatach kompaktowych kadrowanie można wykonać przez niedoskonały wizjer (podatny na zjawisko paralaksy) lub patrząc na ekran, gdzie wyświetlany jest obraz z matrycy. W tym czasie obraz próbkowany jest 15, 30, a nawet 60 razy w ciągu sekundy według schematu: z pomocą światłomierza ustawiane są parametry ekspozycji i próbkowany jest obraz (ładowanie „kondensatorków", opróżnianie, przetwarzanie obrazu, wyświetlanie). Na tej samej zasadzie nagrywane są filmy (pomijamy tutaj kwestie przetwarzania dźwięku) z tą różnicą, że dane są zapisywane w pamięci trwałej. Z technicznego punktu widzenia matryca jest stale oświetlona, i w celu uniknięcia nierównomiernego naświetlenia pojawia się konieczność zastosowania kompensacji - w matrycach CCD dane odczytywane są linia po linii, co trwa, a przecież światło oświetla matrycę cały czas.

Dalsze szczegóły techniczne można znaleźć w [3], [5] i [6]. W kolejnych rozdziałach omówione zostaną mechanizmy kompresji obrazów.

[6]Różni producenci stosują różne rozszerzenia: raw, raf, arw, rwl itd.
[7]Tu efektem będzie plik (na ogół) z rozszerzeniem .jpeg

1.6 Redundancja sygnału

W poprzednim podrozdziale poznaliśmy metodę tworzenia zdjęcia poprzez próbkowanie i zapis w formacie raw. Ten typ danych cechuje pewna nadmiarowość. Generalnie rozróżniamy dwa typy nadmiarowości [1, s. 313]:

1. **statystyczną** - intuicyjnie związana jest z występowaniem w sygnale tych danych, których nie trzeba dostarczać do odbiornika, gdyż sam może je odtworzyć. Ten typ nadmiarowości można usunąć stosując kompresję bezstratną. Istnieje wiele typów redundancji statystycznej, na przykład: przestrzenna[8], czasowa[9], lub kodowania, która ma ścisły związek ze sposobem organizacji danych. Wówczas minimalną średnią ilość danych konieczną do zakodowania symbolu określa **entropia**.

   Zgodnie z [1, s. 314] entropią nazywać będziemy następującą sumę:

   $$H = -\sum_i p_i \; log_2 p_i$$

   gdzie p_i to prawdopodobieństwo wystąpienia i-tego symbolu w sygnale. Zaś kody sygnału charakteryzować będzie średnia liczba bitów:

   $$\overline{b} = \sum_i n_i \; p_i$$

   gdzie n_i to długość kodu i-tego symbolu, a p_i to jego prawdopodobieństwo. Jeśli średnia liczba bitów na symbol jest większa od entropii, to wtedy mamy do czynienia z nadmiarowością w sygnale.

2. **psychowizualną** - związana z obecnością w sygnale informacji mało istotnych dla odbiorcy. Ten rodzaj nadmiarowości usuwa się używając kompresji stratnej.

[8] Gdy wartość w punkcie można z dużym prawdopodobieństwem odgadnąć, odnosząc się do punktów sąsiednich

[9] Związana ze statystyczną korelacją kolejnych klatek materiału wideo.

1.7 Metody kompresji obrazów

Zasadniczo mamy dostępne dwie metody kompresji obrazów: stratną i bezstratną. Bezstratna metoda polega na tym, że obraz jest przetwarzany do postaci zajmującej mniej miejsca, dzięki czemu oszczędzamy miejsce w pamięci. Metoda ta jest w pełni odwracalna i dekompresując sygnał, uzyskujemy obraz pierwotny.

Inaczej jest z kompresją stratną. Algorytmy kompresji stratnej są na ogół bardziej efektywne, niż algorytmy bezstratne, natomiast mają jedną, na pierwszy rzut oka - poważną wadę: tracimy bezpowrotnie informacje. W praktyce umiejętne zastosowanie stratnych algorytmów skutkuje otrzymaniem całkiem niezłego jakościowo obrazu, zajmującego o wiele mniej miejsca.

1.7.1 Kompresja bezstratna

Kompresja bezstratna nazywana jest często kodowaniem. Na ogół proces kompresji podzielony jest na dwa etapy. Pierwszy z nich to modelowanie, w którym tworzona jest pośrednia reprezentacja cech danych źródłowych, do której dopasowuje się metodę binarnego kodowania. Polega ono na użyciu metod statystycznych i predykcyjnych, oszczędnego opisu lokalnych zależności danych, przekształceniu danych w sekwencje o większej podatności na kodowanie binarne, konstrukcji słownika z najczęściej występującymi frazami itd. Modelowanie można zrealizować na trzy zasadnicze sposoby [2, s. 28-29]:

1. Przekształcając dane z przestrzeni oryginalnej w inną przestrzeń, z wykorzystaniem metrycznych zależności danych, określonego sposobu porządkowania danych lub zmiany wymiarowości określonej dziedziny danych, transformacji całkowitoliczbowych itp.

2. Tworząc model deterministyczny opisujący bezpośrednie właściwości danych

3. Opracowując probabilistyczny model źródła informacji (założenie stacjonarności i ergotyczności źródła) na podstawie określonej postaci kontekstu wystąpienia danych

Możliwe jest również łącznie różnych metod celem uzyskania większej podatności na kodowanie. W założeniu pośrednia reprezentacja ma lepsze właściwości statystyczne (większe zróżnicowanie prawdopodobieństw symboli, dłuższe ciągi jednakowych bądź podobnych symboli). Zostaje to wykorzystane w drugim etapie: kodowaniu binarnym, które polega na utworzeniu w możliwie oszczędny sposób sekwencji bitowej jednoznacznie reprezentującej kodowane dane. Oba te etapy mogą się wzajemnie przeplatać. Ponadto modelowanie i kodowanie binarne to funkcje będące bijekcjami, dzięki czemu możliwe jest przeprowadzenie procesu odwrotnego, w celu uzyskania oryginalnego sygnału.

1.7.2 Kompresja stratna

Intuicyjnie rzecz ujmując: kompresja stratna polega na pominięciu informacji, które w znikomy sposób mają wpływ na obiór przez człowieka danych jako całości – bazują na fizjologicznych aspektach postrzegania rzeczywistości przez nasz mózg. Wykorzystują również niedoskonałości urządzeń, którymi te dane odtwarzamy (ograniczona dokładność wyświetlaczy, przetworników cyfrowo-analogowych itp.).

Kompresję osiąga się poprzez: usunięcie bardzo wysokich lub bardzo niskich dźwięków (praktycznie niesłyszalnych lub w ogóle niesłyszalnych) w utworze, ograniczeniu liczby możliwych wartości na każdy piksel obrazu – tzw. kwantyzacja, ograniczenie szczegółowości, zwłaszcza w dynamicznych scenach filmowych, ograniczenie liczby kolorów. Są to procesy nieodwracalne i sygnał poddany procesowi kompresji stratnej jest jedynie przybliżeniem sygnału oryginalnego.

Metody te są zadowalające w przypadku typowego użytkowania tych danych, przez statystycznego użytkownika. Użytkownicy sprzętu z segmentu hi-end najczęściej korzystają z danych słabo kompresowanych, lub w ogóle niepoddawanych kompresji stratnej, o większej dynamice i większych rozdzielczościach. Również do zastosowań profesjonalnych unika się stosowania kompresji stratnej, przynajmniej na etapie montażu.

2 Kody Huffmana

Kody Huffmana to bardzo często wykorzystywana metoda kompresji bezstratnej. Wykorzystywane m.in. w standardach JPEG i MPEG-2.

Kody Huffmana powstają poprzez przypisanie symbolom, słów kodowych o różnej długości. Sedno metody opiera się na tym, aby przypisywać najkrótsze słowa symbolom, których prawdopodobieństwo wystąpienia jest największe, zaś symbolom „rzadszym" - dłuższe. Ponieważ owe przypisanie jest jednoznaczne, łatwo jest ten proces odwrócić. Dodatkowo nie jest pomijana żadna część sygnału. Dzięki temu sygnał można bezstratnie kompresować kodami Huffmana.

Rozróżniamy dwa typy kodów Huffmana: statyczne i dynamiczne. Oba typy omówimy w kolejnych podrozdziałach.

2.1 Statyczne kody Huffmana

Sedno tej metody polega na tym, że tworzona jest jedna tabela kodów do całego sygnału, a następnie jest on w całości kompresowany tymi kodami.

Aby móc skonstruować optymalne[10] kody, musimy znać statystykę sygnału. W związku z tym nasz sygnał (obraz) trzeba przeczytać dwukrotnie: pierwszy raz w celu oznaczenia prawdopodobieństw symboli i stworzenia odpowiedniego słownika, a drugi raz w celu zakodowania sygnału. W efekcie dostajemy skompresowany sygnał z dołączonym słownikiem, zaś sam proces nazywamy adaptacyjnym kodowaniem Huffmana [1, s. 336] - nie czynimy żadnych wstępnych założeń względem statystyki sygnału.

Naturalnym pytaniem jest: czy konieczność dołączenia słownika nie spowoduje, że skompresowany obraz będzie zajmował tyle samo lub nawet więcej pamięci, niż oryginalny obraz? Otóż tak się w istocie może zdarzyć dla odpo-

[10]Optymalne w sensie naszego podejścia, czyli cały sygnał kompresujemy jedną tabelą kodów.

wiednio dobranych obrazów. Natomiast w przypadku statystycznego obrazka do takich sytuacji nie dochodzi.

Tablice kodów można również ustalić odgórnie z pomocą metod statystycznych, i umieścić je na stałe w koderze. Wówczas dane są odczytywane tylko raz – w celu zakodowania. Wadą takiego rozwiązania jest zróżnicowana efektywność – spada dla sygnałów, których statystyka odbiega od założonej, a dla odpowiednio dobranych przypadków znowu możemy dostać na wyjściu sygnał o większym rozmiarze, niż oryginał, pomimo tego, że nie zawiera on słownika. Kolejną wadą takiego rozwiązania jest fakt, że dekoder musi mieć tablice, którymi kodowany był sygnał, co wymusza definiowanie sztywnych standardów kodowania.

Metody adaptacyjnej używa się najczęściej do obrazów statycznych. Natomiast druga metoda najczęściej stosowana jest w przypadku materiałów wideo [1, s. 336] – tam kodami Huffmana koduje się błędy predykcji [1, s. 337].

2.1.1 Ogólny algorytm

Przedstawimy teraz podstawowy algorytm projektowania kodów Huffmana zaczerpnięty z [1, s. 337]:

1. Kodowaniu podlega alfabet składający się z N różnych symboli s_1, s_2, s_3, ..., s_N. Należy wyznaczyć prawdopodobieństwa występowania poszczególnych symboli wynoszące odpowiednio p_1, p_2, p_3, ..., p_N.

2. Dwa symbole o najmniejszych prawdopodobieństwach występowania zastępuje się nowym symbolem S_j, $j = 1$. Załóżmy że tymi symbolami są s_{N-1} i s_N. Wtedy nowy alfabet będzie się składał z symboli s_1, s_2, s_3, ..., s_{N-2}, S_i, a prawdopodobieństwo skojarzone z S_j wyniesie $p_{N-1} + p_N$.

3. Krok 2 jest powtarzany dla kolejnych wartości j tak długo, aż alfabet składał się będzie z dokładnie jednego symbolu.

W ten sposób budujemy drzewo kodu, a przypisując zera i jedynki do odpowiednich gałęzi, wyznacza się słowa kodowe dla każdego symbolu.

Powyższa metoda gwarantuje jednoznaczność kodowania – zapewnia że żadne wygenerowane słowo kodowe nie jest przedrostkiem drugiego. Na przykład niejednoznaczność istniałaby wtedy gdybyśmy otrzymali słowa kodowe: 10 i 101. Prowadziłoby to do niejednoznaczności w chwili dekodowania.

Może się zdarzyć, że w chwili budowania drzewa trafimy na dwa symbole, które mają takie same prawdopodobieństwa. W takim przypadku wybór jest dowolny – w zależności od wyboru otrzymamy różne kody, ale średnia liczba bitów wymagana do zakodowania jednego symbolu będzie we wszystkich przypadkach identyczna [1, s. 337].

Przykład 2.1. Weźmy alfabet $\sum = \{A, B, C, D, E, F, G, H\}$ oraz słowo długości 100:

```
CCFCCBCFCBCBCACGCCBCGACCCDGCAGECCGCCEEECEEGCACEAGC
BACEGCEHCAGACHCCGFCCHCCGGCHCGFCCHCCGCHCGFCCHBCCFGC
```

Zliczymy teraz częstości występowania znaków komendą:

```
echo "<słowo>"| grep -o . | sort | uniq -c | sort -rn
```

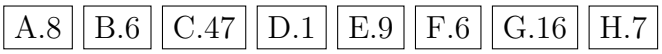
Otrzymując:

```
47 C
16 G
9 E
8 A
7 H
6 F
6 B
1 D
```

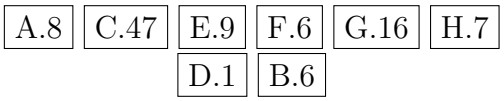
Określamy wagi każdego z elementów (tu użyte częstości):

Znak	A	B	C	D	E	F	G	H
Waga:	8	6	47	1	9	6	16	7

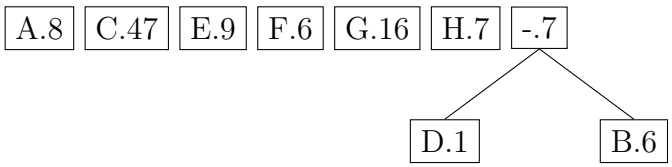
Każdy symbol umieszczamy w oddzielnym węźle wraz z wagami:



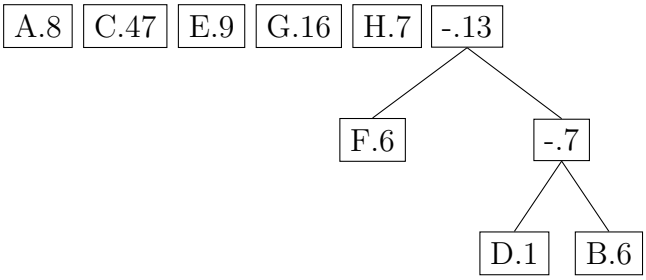
Z listy węzłów wybieramy dwa, które mają najmniejsze wagi:

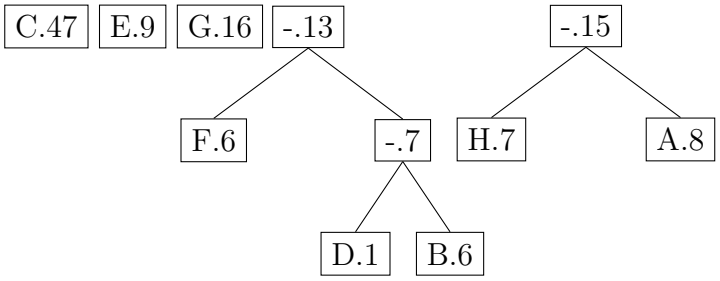


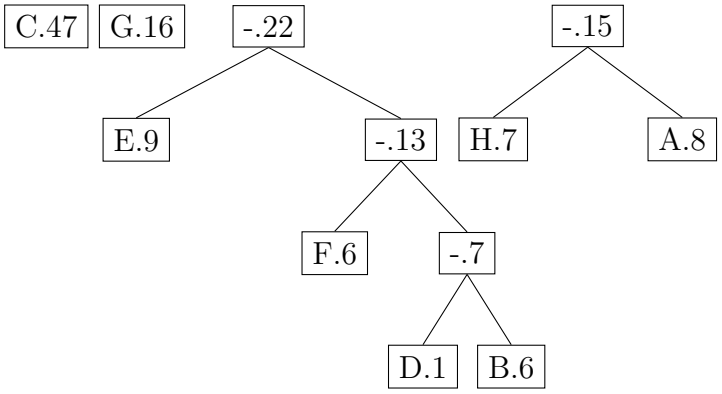
Następnie konstruujemy z nich drzewo takie, że stają się liśćmi, a korzeniem jest nowy węzeł z wagą równą sumie wag jego liści, i traktujemy go jako "zwykły" węzeł:

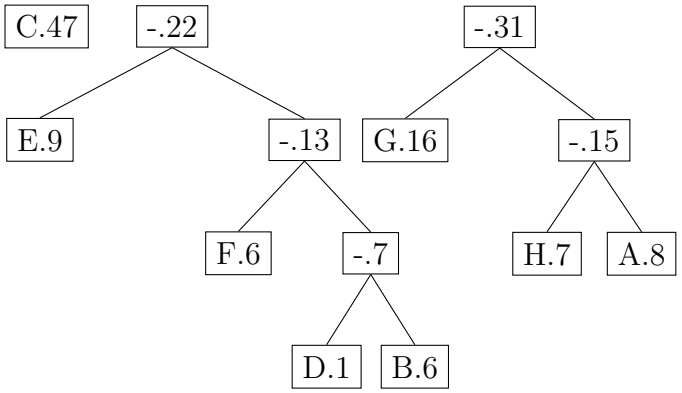


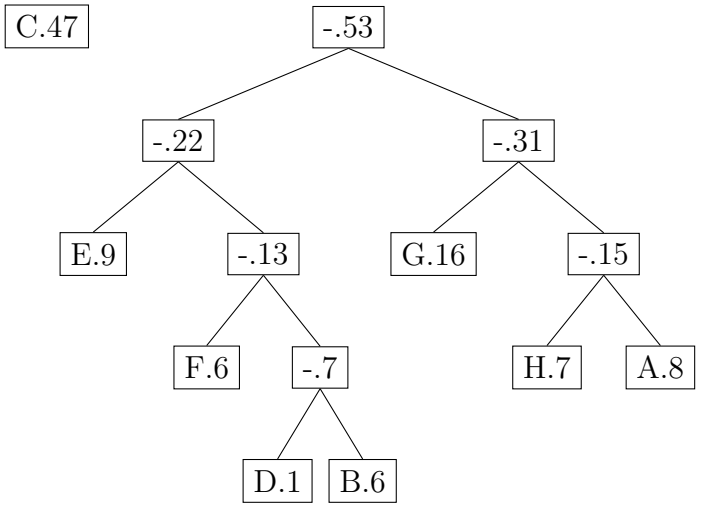
Kontynuujemy ten proces, aż do scalenia wszystkiego w jedno drzewo:



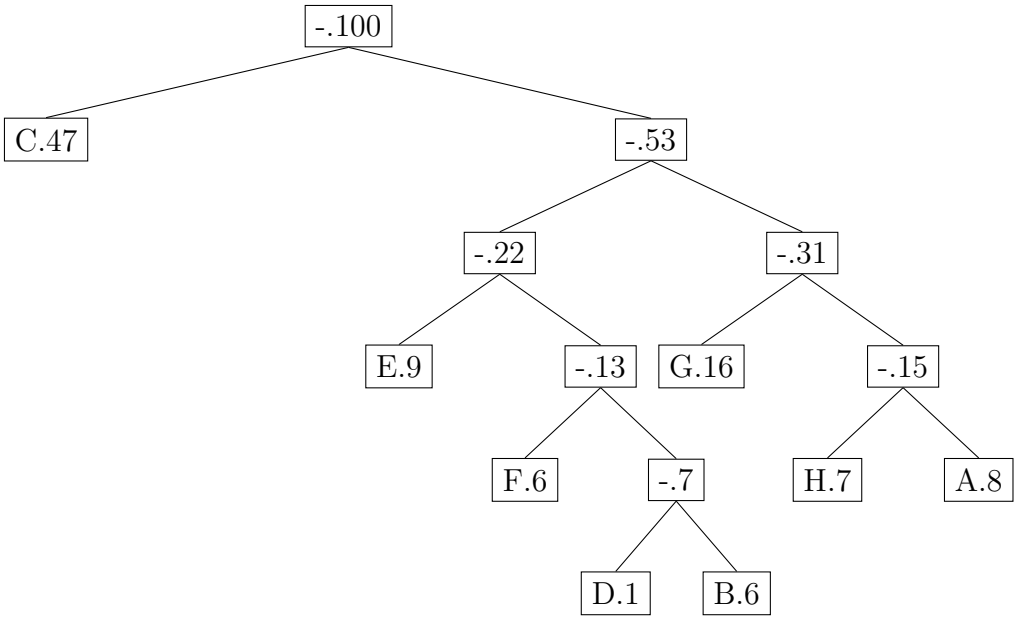

C.47
E.9
G.16
-.13
-.15
F.6
-.7
H.7
A.8
D.1
B.6


C.47
G.16
-.22
-.15
E.9
-.13
H.7
A.8
F.6
-.7
D.1
B.6

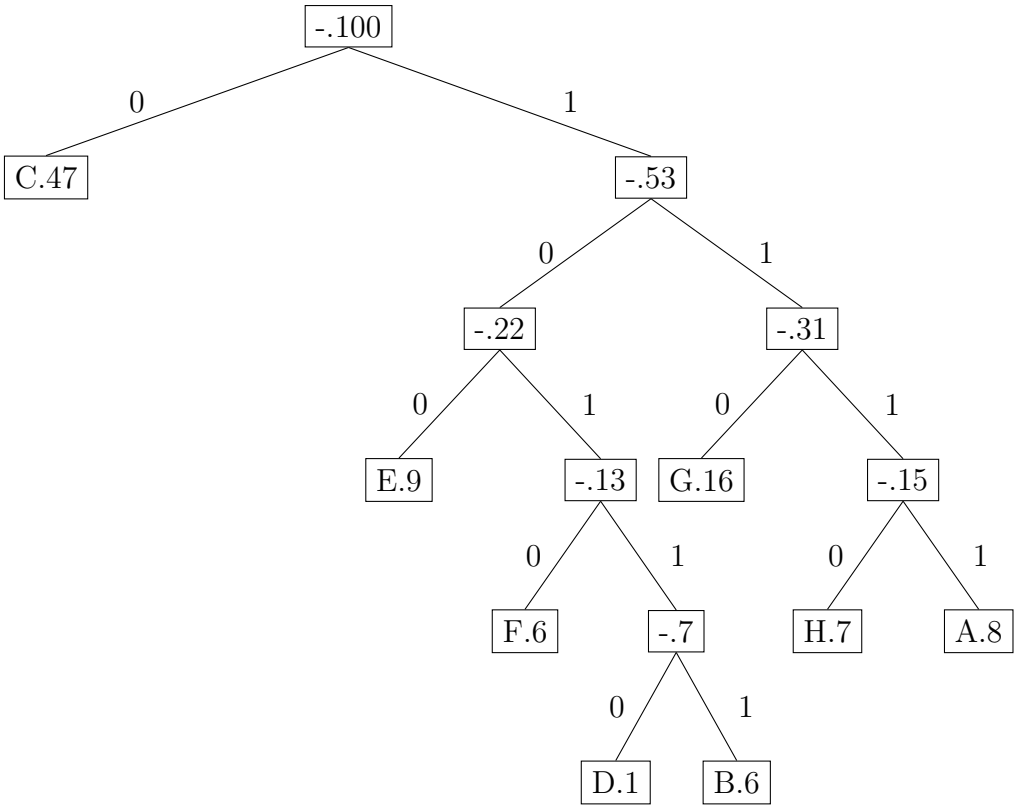

C.47
-.22
-.31
E.9
-.13
G.16
-.15
F.6
-.7
H.7
A.8
D.1
B.6


C.47
-.53
-.22
-.31
E.9
-.13
G.16
-.15
F.6
-.7
H.7
A.8
D.1
B.6

Końcowe drzewo to:



Oznaczamy odpowiednio każdą krawędź drzewa:



A następnie odczytujemy kody dla każdego symbolu:

Symbol	Kod
A	1111
B	10111
C	0
D	10110
E	100
F	1010
G	110
H	1110

I stąd mamy zakodowane słowo:

```
001010001011101010010111010111011111011000101110110111100010110
100111111010000110001001001000100100110011110100111111001011111
110100110010011100111111011110111000110101000111000110110011100
110101000111000110011100110101000111010111001010110 0
```

otrzymując kod długości 241 bitów.

Sprawdźmy ile zyskaliśmy kodami Huffmana: zastosujemy prosty kod binarny postaci:

Symbol	Kod
A	000
B	001
C	010
D	011
E	100
F	101
G	110
H	111

Zakodowane słowo wygląda następująco:

```
010010101010010001010101010001010001010000010110010010001010110
000010010010011111001000011010001001011001001010010010001010010 0
110010000010100000110010001000010100110010100111010000110000010
111010010110101010010111010010110110010111010110101010010111010
010110010111010110101010010111001010010101110010
```

I jest ono długości 300 bitów.

Różnica w długościach słów wynosi 59 bitów. Współczynnik kompresji[11] w tym przypadku wynosi: $CR = \frac{241}{300} = 0,80(3)$

[11]Drobne oszustwo - nie uwzględniam tutaj słownika kodów Huffmana.

2.1.2 Dekodowanie sygnału

Dekodowanie sygnału zakodowanego kodami Huffmana można przeprowadzić dwiema metodami:

1. Organizując kody w strukturę drzewiastą - w ten sposób odtwarzamy sygnał symbol po symbolu

2. Używając tablicy kodów. Wyszukujemy najdłuższy kod w tablicy i obliczamy jego długość. Następnie pobieramy ciąg takiej właśnie długości, i szukamy w tablicy kodu, który rozpoczyna nasz ciąg. Gdy go znajdziemy, zapisujemy zdekodowany symbol, usuwamy z ciągu jego kod, a następnie uzupełniamy ciąg do pierwotnej długości. Procedurę tę powtarzamy, aż cały sygnał zostanie zdekodowany.

Naturalny problem, jaki się pojawia, to problem stopu: kiedy zatrzymać dekodowanie sygnału. Oczywiście w chwili wystąpienia błędu - brak kodu w słowniku, problemy z pamięcią itp. Jeśli wykluczymy już takie elementy, to pozostaje poinformować dekoder o zakończeniu sygnału. Można to zrobić na kilka sposobów: dopisując na koniec do sygnału symbol nie istniejący w dotychczasowym alfabecie, informując dekoder o długości sygnału itp.

Kolejnym problemem mogą okazać się długości kodów. Kody Huffmana potrafią być bardzo długie, co powoduje wydłużenie procedury dekodowania. Opisana druga metoda - tablicowa, jeśli niewłaściwie zastosowana[12], to jest bardzo wrażliwa na taki problem. Oczywiście dla odpowiednio dobranych przypadków[13] uzyskamy kody równej długości, które ułożą się w ładne[14] drzewo, czasem nawet zrównoważone, i czas wyszukania w takim drzewie będzie logarytmiczny względem liczby kodów (lub prawie logarytmiczny), ale w praktyce te drzewa na szczęście nie są zrównoważone.

Dlatego czasami używa się modyfikacji kodów Huffmana, polegającej na ograniczeniu ich długości. Ma to oczywiście negatywny wpływ na stopień kom-

[12]Czyli tablica posortowana względem kodów, a nie ich prawdopodobieństw

[13]Takich, w których prawdopodobieństwo wystąpienia każdego symbolu jest takie samo, lub prawie takie samo

[14]Ale niepraktyczne - dla zrównoważonych sygnałów uzyskuje się słaby stopień kompresji

presji, ale za to przyspiesza sam proces kompresji i dekompresji [1, s. 340].

W najprostszym przypadku kodowaną liczbę K przedstawia się w postaci:

$$K = l * L + m, \qquad m = KmodL$$

L to pewna wybrana liczba, l to wynik dzielenia bez reszty K przez L, zaś liczby l i m są niezależnie kodowane według różnych tablic kodów [1, s. 340].

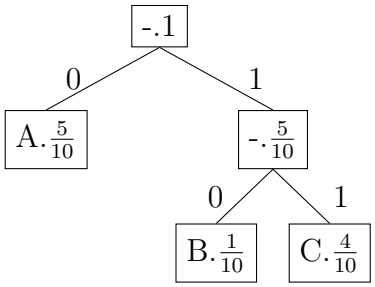
W kompresji materiałów wideo stosuje się inną „sztuczkę": symbolom o bardzo małych prawdopodobieństwach wystąpienia (a więc tym o najdłuższych kodach) przypisuje się jeden wspólny ciąg bitowy (zwykle określany jako ESCAPE), po którym przesyła się sygnał z pomocą kodów o stałej długości słowa kodowego [1, s. 340].

2.1.3 Kodowanie blokowe

Do tej pory rozważaliśmy adaptacyjne kody Huffmana, których cechą charakterystyczną było generowanie słów kodowych o optymalnej długości, takich że średnia długość słowa kodowego dla każdego symbolu była minimalna. Następnie symbol po symbolu kodowaliśmy wygenerowanymi kodami. A gdyby tak kodować nie pojedyncze symbole, tylko całe ich grupy?

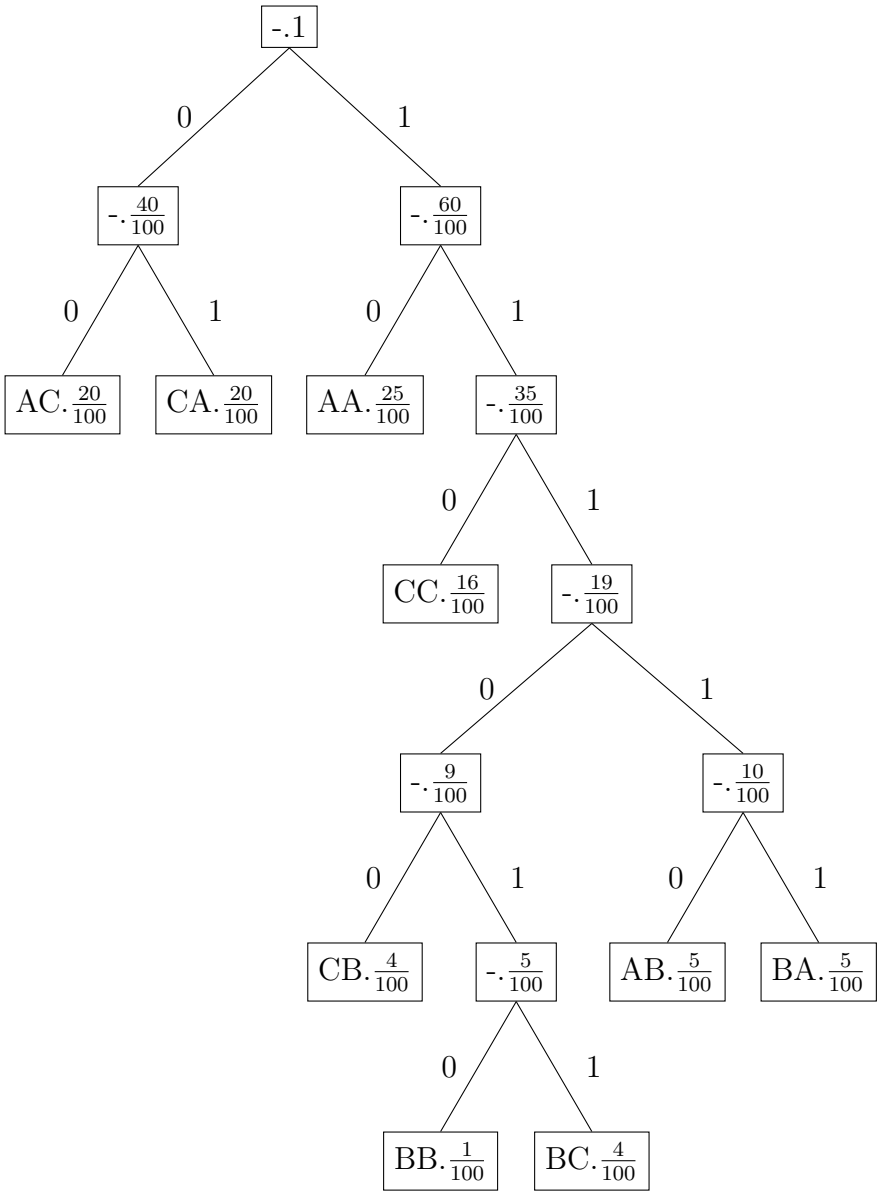
Taka modyfikacja kodów Huffmana jest również spotykana w praktyce i nazywa się je blokowymi kodami Huffmana. Modyfikacja ta poprawia współczynnik kompresji, kosztem komplikacji kodera i dekodera.

Przykład 2.2. Sprawdźmy działanie metody blokowej w praktyce. Niech $\sum = \{A, B, C\}$ będzie alfabetem naszego sygnału. Prawdopodobieństwa symboli: $P(A) = \frac{5}{10}$, $P(B) = \frac{1}{10}$, $P(C) = \frac{4}{10}$. Konstruujemy drzewo:



Średnia długość kodu na każdy symbol to $1,5$ bita.

Niech teraz długość bloku jest równa 2. Drzewo kodów wygląda następująco:



Stąd średnia długość słowa kodowego dla każdego bloku to $2,78$ bita, co daje średnio $1,39$ bita na pojedynczy symbol.

Zwiększmy teraz długość bloku do trzech symboli. Rozkład prawdopodobieństw i przydzielone kody obrazuje poniższa tabela:

Blok	Prawdopodobieństwo	Kod Huffmana
AAA	0,125	110
AAB	0,025	101111
AAC	0,1	000
ABA	0,025	01100
ABB	0,005	0111110
ABC	0,02	111110
ACA	0,1	001
ACB	0,02	111100
ACC	0,08	1000
BAA	0,025	01101
BAB	0,005	0111111
BAC	0,02	111101
BBA	0,005	11111110
BBB	0,001	111111110
BBC	0,004	111111111
BCA	0,02	101100
BCB	0,004	11111100
BCC	0,016	011110
CAA	0,1	010
CAB	0,02	101101
CAC	0,08	1001
CBA	0,02	101110
CBB	0,004	11111101
CBC	0,016	011100
CCA	0,08	1010
CCB	0,016	011101
CCC	0,064	1110

Stąd średnia długość słowa kodowego dla każdego bloku to $4,11$ bita, co daje średnio $1,37$ bita na pojedynczy symbol.

Podsumujmy:

Dł. bloku	L. kodów	Gł. drzewa	Śr. dł. kodu (na sym.)	Entropia
1	3	2	1,5 (1,5)	1,36 (1,36)
2	9	6	2,78 (1,39)	2,72 (1,36)
3	27	9	4,11 (1,37)	4,08 (1,36)

Dalsze zwiększanie długości bloków przyniosłoby w tym przypadku jeszcze słabsze efekty, komplikując tylko kodowanie i dekodowanie.

2.2 Dynamiczne kody Huffmana

W poprzedniej metodzie, aby zbudować listę kodów, musieliśmy znać statystykę sygnału. Natomiast dynamiczna metoda buduje drzewo kodów „w locie" - w miarę napływania danych koder poprawia dotychczas utworzone drzewo kodów. Zatem statystyka całego sygnału jest nieznana - znamy jedynie statystykę sygnału, który został odebrany. Zaletą tej metody jest fakt, że nie ma potrzeby przesyłania słownika kodów. Wadą jest to, że i koder, i dekoder muszą wykonać mniej więcej tę samą pracę: budowa i poprawianie drzewa kodów, co komplikuje budowę dekodera.

2.2.1 Algorytm

Zakładamy początkowy alfabet, każdy symbol z niego ma na początku taką samą wagę (np. równą 1)[15]. W oparciu o nie ustalamy postać początkową drzewa kodów. Zarówno koder, jak i dekoder mają to samo drzewo początkowe.

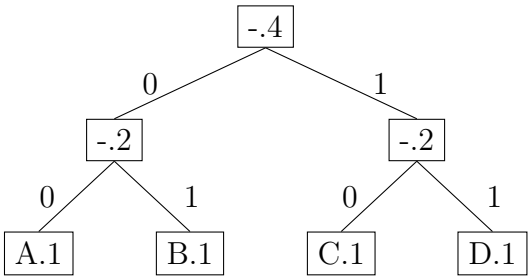
Koder odczytuje z wejścia symbol, koduje go z użyciem bieżącego drzewa, a następnie je aktualizuje. Są różne algorytmy (np. algorytm 3.4 z [2, s. 83] oparty o wagi i numerowanie wierzchołków).

[15]Jeśli znamy statystykę sygnału, to drzewo początkowe dostosowujemy do niej.

My dla uproszczenia założymy, że zwiększana jest waga wierzchołka i w razie potrzeby odpowiednie poddrzewa są ze sobą zamieniane. Zakładamy ponadto, że dla dwóch potomków węzła, lewy ma co najmniej taką samą wagę jak prawy, oraz waga wierzchołka na poziomie wyższym, jest co najmniej taka sama, jak waga wierzchołka na poziomie niższym.

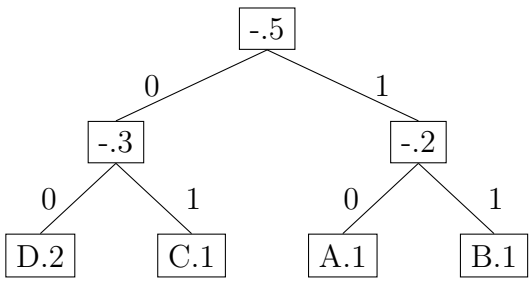
Dekodowanie odbywa się analogicznie. Na wejściu mamy strumień bitów. Odczytujemy sygnał bit po bicie, jednocześnie przesuwając się po drzewie, aż dotrzemy do węzła zawierającego symbol. Zapisujemy go i aktualizujemy drzewo według takich samych reguł, jak w przypadku kodera.

Przykład 2.3. Niech $\sum = \{A, B, C, D\}$ będzie alfabetem naszego sygnału, a koder i dekoder mają ustalone drzewo początkowe:

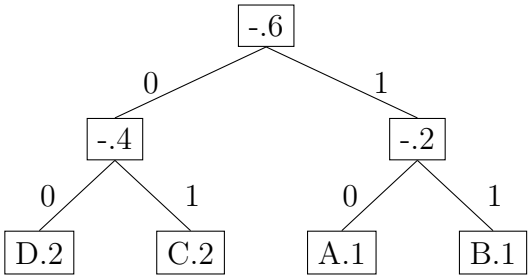


Skompresujmy sygnał DCBDDCDD z pomocą dynamicznych kodów Huffmana.

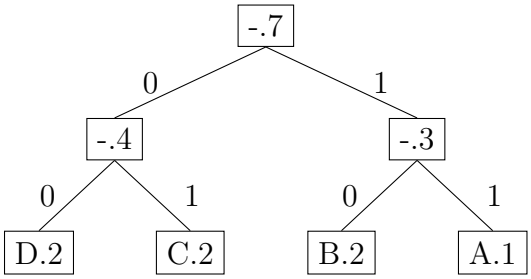
1. Kodujemy D: **11** i aktualizujemy drzewo do postaci:



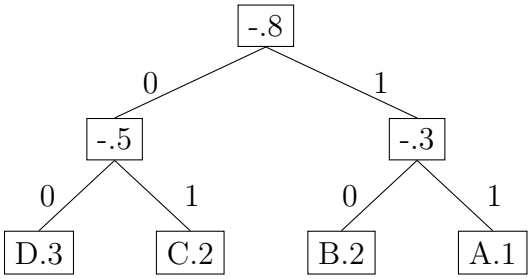
2. Kodujemy C: **01** i aktualizujemy drzewo do postaci:



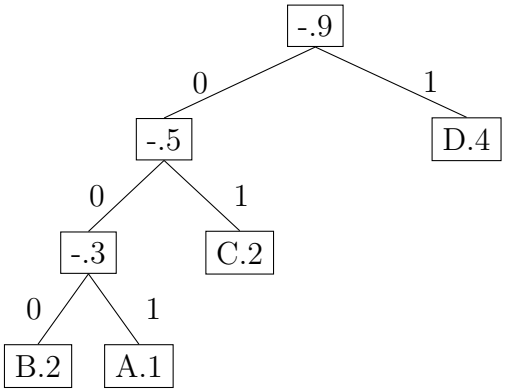
3. Kodujemy B: **11** i aktualizujemy drzewo do postaci:



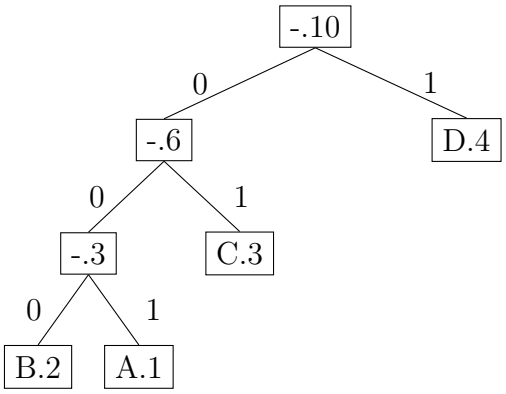
4. Kodujemy D: **00** i aktualizujemy drzewo do postaci:



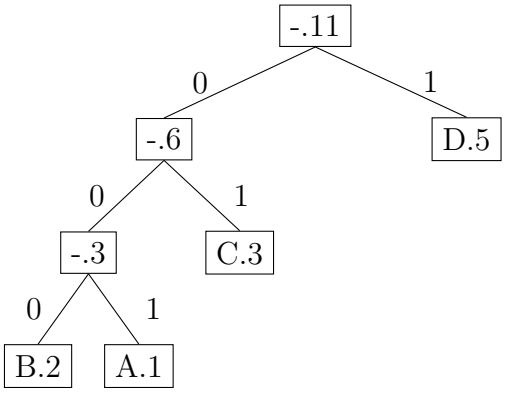
5. Kodujemy D: **00** i aktualizujemy drzewo do postaci:



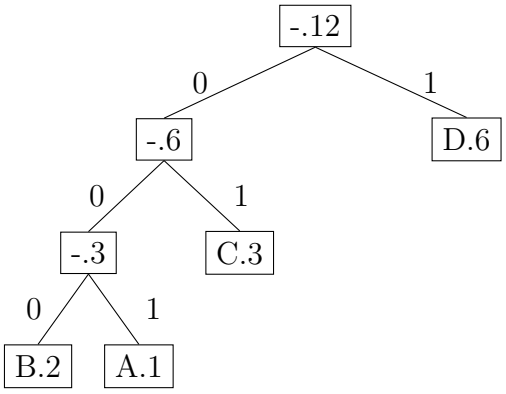
6. Kodujemy C: **01** i aktualizujemy drzewo do postaci:



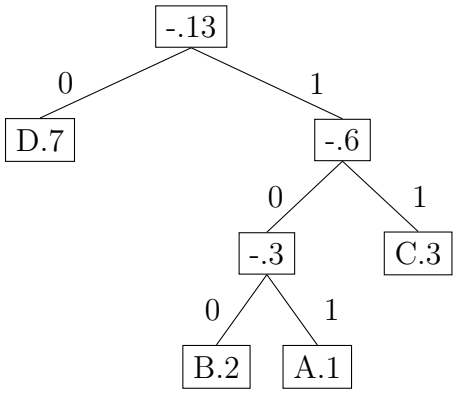
7. Kodujemy D: **1** i aktualizujemy drzewo do postaci:



8. Kodujemy D: **1** i aktualizujemy drzewo do postaci:



9. Kodujemy D: **1** i aktualizujemy drzewo do postaci:



Zatem zakodowany ciąg DCBDDCDD wygląda następująco:

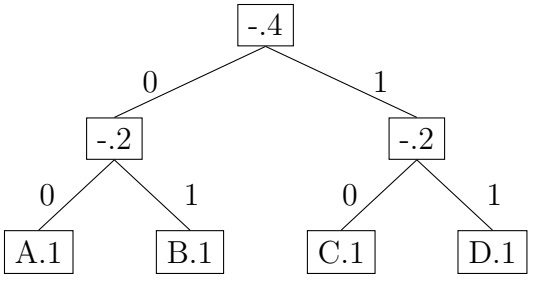
110111000001111

i ma długość 15 bitów.

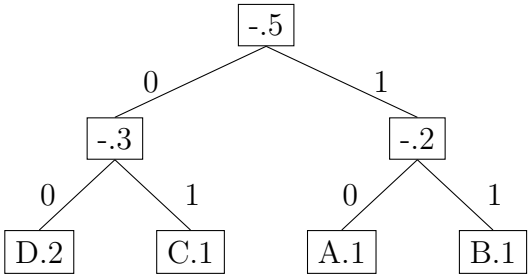
Stosując kod binarny (A: 001, B: 010, C: 100, D: 101) uzyskalibyśmy zakodowany sygnał postaci: 101100010101101100101101 o długości 24 bitów.

Rozkodujmy teraz sygnał: 110111000001111.

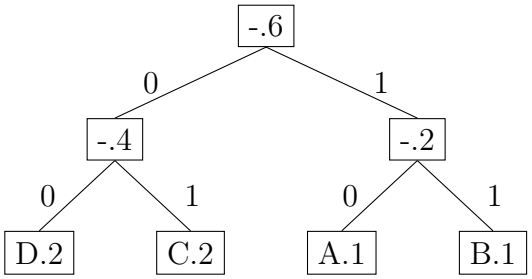
Dane mamy drzewo początkowe:



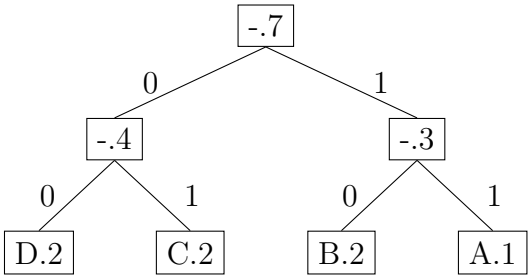
Odczytujemy symbol: **D** i aktualizujemy drzewo do postaci:



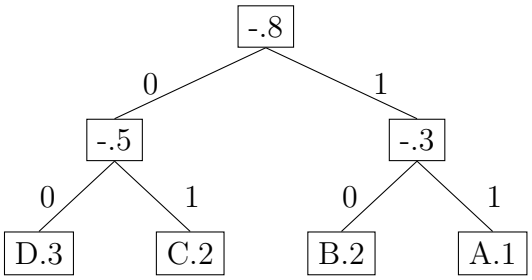
Odczytujemy symbol: **C** i aktualizujemy drzewo do postaci:



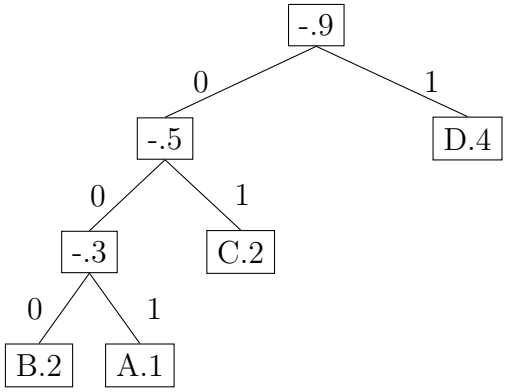
Odczytujemy symbol: **B** i aktualizujemy drzewo do postaci:



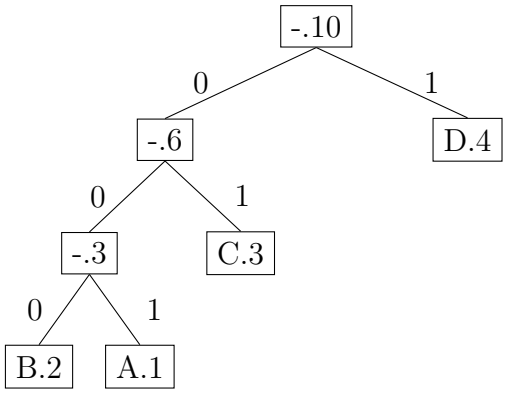
Odczytujemy symbol: **D** i aktualizujemy drzewo do postaci:



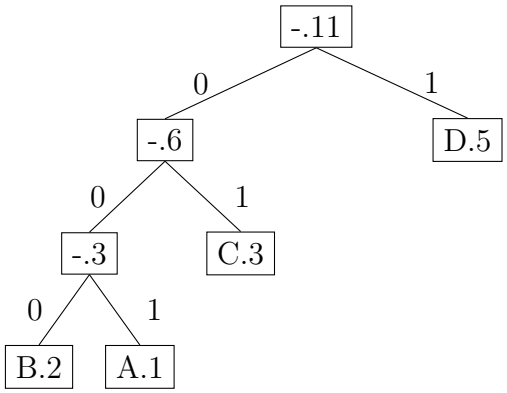
Odczytujemy symbol: **D** i aktualizujemy drzewo do postaci:



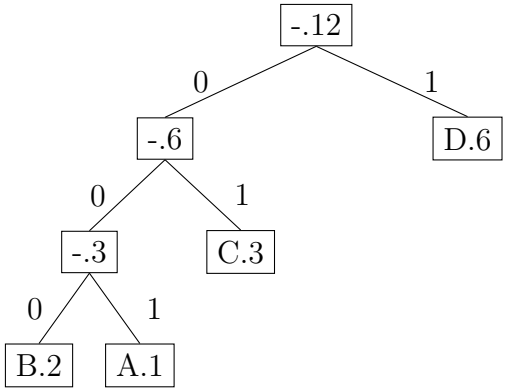
Odczytujemy symbol: **C** i aktualizujemy drzewo do postaci:



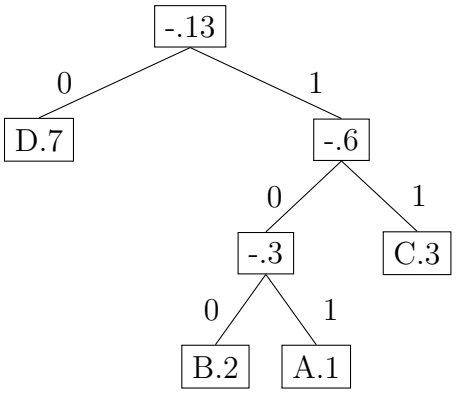
Odczytujemy symbol: **D** i aktualizujemy drzewo do postaci:



Odczytujemy symbol: **D** i aktualizujemy drzewo do postaci:



Odczytujemy symbol: **D** i aktualizujemy drzewo do postaci:



Rozkodowaliśmy zatem sygnał DCBDDCDD.

3 Kodowanie arytmetyczne

W kodowaniu arytmetycznym słowa kodowe o różnej długości przypisywane są do ciągów symboli o zmiennej długości, co jest zasadniczą różnicą między kodami Huffmana i kodami arytmetycznymi [1, s. 344]. Ponadto kodami arytmetycznymi jesteśmy w stanie przekroczyć barierę 1 bit/symbol, stanowiącą ograniczenie kodów Huffmana. W kodowaniu arytmetycznym, tak samo, jak w kodach Huffmana rozróżniamy metody statyczne i dynamiczne.

3.1 Algorytm statyczny

Dane mamy słowo składające się z symboli.

1. Na początku zliczamy prawdopodobieństwa wystąpienia każdego symbolu w słowie i proporcjonalnie dzielimy odcinek $[0, 1)$ na przedziały lewostronnie domknięte.

2. Kodujemy pierwszy symbol, czyli wybieramy przedział przydzielony odcinkowi

3. Kodujemy następny symbol: uprzednio wybrany odcinek dzielimy proporcjonalnie do prawdopodobieństwa wystąpienia każdego z symboli, analogicznie jak na początku, i wybieramy przedział, do którego należy kodowany symbol.

4. Krok trzeci powtarzamy, aż zakodujemy cały ciąg symboli.

5. Uzyskany przedział jednoznacznie określa cały ciąg symboli. Wybieramy teraz z przedziału tę liczbę, której postać binarna jest najkrótsza. To ona, razem z informacją na temat prawdopodobieństw będzie teraz stanowiła jednoznaczny kod całego ciągu.

Przykład 3.1. Prześledźmy pracę kodera i dekodera na przykładzie ciągu:

AECDEBA

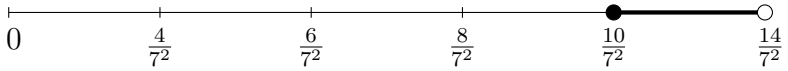
Oznaczamy prawdopodobieństwa i przydzielamy przedziały:

Symbol	Prawdopodobieństwo	Przedział
A	$\frac{2}{7}$	$[0, \frac{2}{7})$
B	$\frac{1}{7}$	$[\frac{2}{7}, \frac{3}{7})$
C	$\frac{1}{7}$	$[\frac{3}{7}, \frac{4}{7})$
D	$\frac{1}{7}$	$[\frac{4}{7}, \frac{5}{7})$
E	$\frac{2}{7}$	$[\frac{5}{7}, 1)$

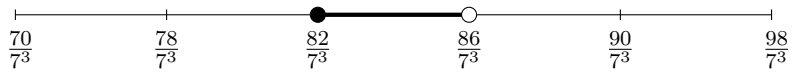
Zakładamy, że kodujemy bloki długości 7 (w związku z tym nie ma potrzeby przesyłania długości bloku).

Kodujemy symbol A:

Kodujemy symbol E:



Kodujemy symbol C:

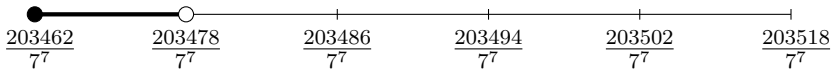


Kodujemy symbol D:

Kodujemy symbol E:

Kodujemy symbol B:

Kodujemy symbol A:



Zatem liczba powinna należeć do przedziału $[\frac{203462}{7^7}, \frac{203478}{7^7})$. Łatwo sprawdzić, że liczba $\frac{253}{1024}$ należy do niego, i jest tą, która ma najkrótszą postać binarną. Załóżmy, że liczbę przesyłamy w postaci: $a_1 \cdot 2^0 + a_2 \cdot 2^{-1} + + a_3 \cdot 2^{-2} + a_4 \cdot 2^{-3} \cdot \ldots$. Wtedy nasza liczba będzie miała długość 11 bitów, i będzie postaci:

000 111 111 01

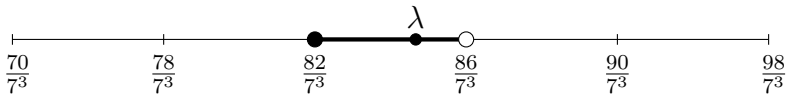
Zakodowanie słowa prostymi kodami binarnymi zajmie 21 bitów, a statycznymi kodami Huffmana 16 bitów.

Rozkodujmy teraz sygnał. Dekoder otrzymuje liczbę 00011111101 i wie, że definiuje ona blok siedmiu symboli. Rozkodujmy więc tę informację. W pierwszej kolejności konwertujemy liczbę do postaci dziesiętnej, otrzymując liczbę $\frac{253}{1024} \approx 0,2470703125$. Nazwijmy ją liczbą λ. Następnie postępujemy analogicznie jak w przypadku kodowania.

λ należy do pierwszego odcinka: odczytujemy symbol A:

Następnie symbol E:

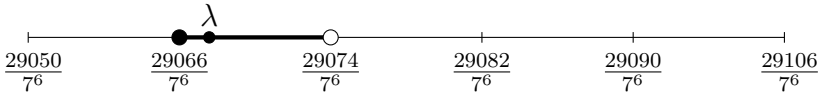
Symbol C:



Symbol D:

Symbol E:

Symbol B:



Symbol A:

W efekcie zdekodowaliśmy ciąg:

AECDEBA

3.2 Algorytmy adaptacyjne

Tak samo, jak w przypadku kodów Huffmana, kody arytmetyczne mają swoje warianty adaptacyjne, których realizację można ująć na trzy sposoby:

1. Dowolna długość bloków danych (poprzednio zakładaliśmy stałą długość bloków) - w tym przypadku do liczby konieczne jest dołączenie długości bloku aktualnie kodowanego lub ustalenie kiedy zakończyć proces dekodowania.

2. Zmienna statystyka symboli po zakodowaniu bloku - konieczne jest zaimplementowanie analogicznych algorytmów w koderze i dekoderze.

3. Zmienna statystyka po zakodowaniu każdego symbolu - tu również konieczne jest zaimplementowanie analogicznych algorytmów w koderze i dekoderze.

W adaptacyjnych algorytmach stosuje się jeden wybrany sposób lub połączenie dwóch: 1. z 3. lub 1. z 2.

4 Kodowanie transformatowe

Zarówno kody arytmetyczne, jak i Huffmana, to metody kompresji bezstratnej, które symbolom lub ich grupom przypisywały określone kody. W tym rozdziale nakreślimy metody kompresji stratnej, którą można znacząco zmniejszyć rozmiar wynikowego pliku, kosztem nieznacznego pogorszenia jakości.

4.1 Podstawy kodowania transformatowego

Aby dobrze opisać czym jest kodowanie transformatowe i jego motywacje, należy zacząć od podstaw, czyli od tego czym jest oko - technicznie rzecz ujmując jest to narząd służący człowiekowi do próbkowania obrazu. Ludzkie oko jest zdolne do percepcji promieniowania elektromagnetycznego z zakresu częstotliwości od około 390 THz do około 830 THz, co odpowiada długościom fal 380nm - 760nm. Zakres ten nazywamy światłem widzialnym.

Światło wpadając do oka przechodzi (w dużym uproszczeniu) przez rogówkę, źrenicę (fizjologiczny otwór przysłony - tęczówki), soczewkę, mającą zdolność zmiany ogniskowej, ale przede wszystkim skupiającą wiązkę światła na siatkówce - elemencie światłoczułym. Siatkówka z kolei składa się według szacunków z około 130 milionów punktów [1, s. 151]: 120 mln. pręcików - bardzo czułych, odpowiedzialnych za widzenie nocne, i około 6 mln. czopków skupionych głównie w plamce żółtej, które odpowiadają za widzenie barw (dzienne). Warto podkreślić, że oko ma ograniczoną rozdzielczość.

Na przestrzeni lat próbowano sparametryzować jakoś pracę oka. W tym celu tworzono rozmaite przestrzenie barw, czyli matematyczne ujęcia systemu wzrokowego człowieka, próbujące uwzględniać jego fizjologiczne aspekty i ograniczenia. Przykładami przestrzeni barw jest: RGB, CMYK, YPbPr, YCbCr itd. Każda z nich powstała do innych zastosowań: CMYK w poligrafii, YPbPr do transmisji analogowego sygnału wideo itd. Zaznaczyć należy, że stanowią one uproszczenie, zaś nie odwzorowują idealnie całego zakresu światła widzialnego.

Mając powyższe na względzie, możemy teraz wrócić do kodowania transformatowego. Najczęściej stosowaną przestrzenią barw jest RGB, i taki też sygnał uzyskujemy z matryc światłoczułych. W praktyce przy kodowaniu transformatowym nie używa się RGB (R - składowa czerwona, G - składowa zielona, B - składowa niebieska), tylko przestrzeni YCbCr, gdzie mamy Y czyli luminancję, oraz dwie chrominancje: Cb to składowa różnicowa stanowiąca różnicę między luminancją, a składową niebieską (Y-B), zaś Cr to składowa różnicowa stanowiąca różnicę między luminancją, a składową czerwoną (Y-R). Składową zieloną wylicza się na podstawie tych trzech wartości. Kodowaniu poddaje się każdą składową z osobna w zasadniczych trzech etapach:

1. Wyznaczenie widma
2. Kwantyzacja próbek widma
3. Kodowanie bezstratne

W dalszej części pracy omówiony zostanie każdy z punktów. Aby to zrobić, niezbędne jest wprowadzenie pojęcia transformaty Fouriera.

4.2 Transformata Fouriera

Do tej pory próbowaliśmy kompresować obrazy wprost. Alternatywą jest wstępne przekształcanie ich do jakiejś innej postaci. Taką postacią mogłoby być na przykład widmo obrazu uzyskane na drodze transformacji Fouriera. Wówczas mielibyśmy do czynienia nie tyle z wartościami sygnału, co z informacją o składowej o danej częstotliwości w wyjściowym sygnale.

Transformatą Fouriera funkcji całkowalnej $f \in \mathbb{R}$ nazywamy funkcję postaci:

$$\widehat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\ e^{-ix\xi}\ dx \tag{1}$$

Transformację Fouriera da się w pełni odwrócić stosując odwrotną transformatę Fouriera:

$$\widehat{f}(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(\xi)\ e^{ix\xi}\ dx \tag{2}$$

Jest to podstawowy wzór na transformatę Fouriera: dla funkcji zespolonych jednej zmiennej rzeczywistej. Tym czasem nasze sygnały to funkcje dwuwymiarowe dyskretne. Aby obliczyć transformatę takiego sygnału, stosuje się różne modyfikacje transformaty Fouriera. Wprowadźmy na początek jej wielowymiarową modyfikację:

$$\widehat{f}(\xi_1, \ldots, \xi_n) = \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} f(x_1, \ldots, x_n)\ e^{-i(x_1\xi_1 + \ldots + x_n\xi_n)}\ dx_1 \ldots dx_n \tag{3}$$

Stąd dla sygnałów ciągłych dwuwymiarowych transformata ma postać:

$$\widehat{f}(\xi_1, \xi_2) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x_1, x_2)\ e^{-i(x_1\xi_1 + x_2\xi_2)}\ dx_1 dx_2 \tag{4}$$

Wersja dyskretna jednowymiarowa, działająca na okresowych funkcjach zespolonych zmiennej dyskretnej, dla sygnału o N próbkach ma postać:

$$\widehat{g}(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} g(n)\ e^{\frac{-2i\pi nk}{N}}, \qquad dla\ k \in [0, N-1] \tag{5}$$

Należy dodać, że przejście z sygnału ciągłego, na sygnał dyskretny (i odwrotnie), dokonuje się z użyciem zaokrągleń. W związku z tym należy zapewnić odpowiednią dokładność (o czym w dalszej części pracy).

Odwrotne dyskretne przekształcenie definiujemy następująco:

$$g(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} f(k)\ e^{\frac{2i\pi nk}{N}}, \qquad dla\ n \in [0, N-1] \tag{6}$$

Po drobnych modyfikacjach dla $k_1 \in [0, N_1 - 1],\ k_2 \in [0, N_2 - 1]$ otrzymujemy dwuwymiarową wersję dyskretnej transformaty Fouriera, działającą na okresowych funkcjach zespolonych zmiennej dyskretnej:

$$\widehat{g}(k_1, k_2) = \frac{1}{\sqrt{N_1 N_2}} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} g(n_1, n_2)\ e^{\frac{-2i\pi n_1 k_1}{N_1} + \frac{-2i\pi n_2 k_2}{N_2}} \tag{7}$$

I dla $n_1 \in [0, N_1 - 1],\ n_2 \in [0, N_2 - 1]$ odwrotne przekształcenie:

$$g(n_1, n_2) = \frac{1}{\sqrt{N_1 N_2}} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} g(k_1, k_2)\ e^{\frac{-2i\pi n_1 k_1}{N_1} + \frac{-2i\pi n_2 k_2}{N_2}} \tag{8}$$

Dyskretna dwuwymiarowa transformata Fouriera jest przekształceniem separowalnym, co oznacza, że może być rozłożona na obliczenie transformat jednowymiarowych kolumnami i wierszami:

$$\widehat{g}(k_1, k_2) = \frac{1}{\sqrt{N_1 N_2}} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} g(n_1, n_2)\ e^{\frac{-2i\pi n_1 k_1}{N_1} + \frac{-2i\pi n_2 k_2}{N_2}} =$$

$$= \frac{1}{\sqrt{N_2}} \sum_{n_2=0}^{N_2-1} \left(\underbrace{\frac{1}{\sqrt{N_1}} \sum_{n_1=0}^{N_1-1} g(n_1, n_2)\ e^{\frac{-2i\pi n_1 k_1}{N_1}}}_{\widehat{g'}} \right) \cdot e^{\frac{-2i\pi n_2 k_2}{N_2}} \tag{9}$$

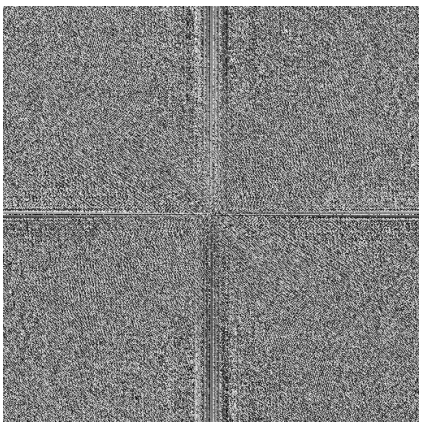
Separowanie ma na celu zmniejszenie liczby operacji koniecznych do wyliczenia transformaty obrazka. Dalsze optymalizacje można osiągnąć stosując algorytm szybkiej transformaty Fouriera (FFT). Bardziej szczegółowe informacje na ten temat można znaleźć w [1, s. 95].

4.2.1 Działanie transformaty Fouriera

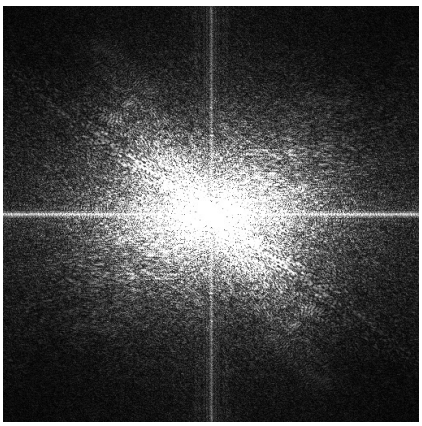
Sprawdźmy w praktyce działanie transformaty Fouriera na obrazku Lena. Na początek policzmy jego transformatę i przedstawmy w postaci obrazka moduł i argument. W tym celu użyjemy programu Matlab oraz matlabowskich funkcji: **fft2** (dwuwymiarowa szybka transformata Fouriera). Funkcja produkuje macierz, w której im bliżej środka, tym współczynniki odpowiadają wyższym częstotliwościom. Aby dostać w centrum próbki o najniższych częstotliwościach użyjemy funkcji **fftshift**, a w celu obliczenia argumentu liczby zespolonej: funkcji **angle**.

Rysunek 11: Oryginalny obrazek Lena.bmp

```
clc;
clear;

[A,map]=imread('Lena.bmp','bmp');
b=fft2(A);
c = fftshift( abs(b) );
d = fftshift( (pi + angle(b))*128/pi );

figure(1);
colormap(map);
image( c/50 );
title('Widmo fazowe');
axis off;
axis equal;

figure(2);
colormap(map);
image( d );
title('Widmo amplitudowe')
axis off;
axis equal;
```



(a) Widmo fazowe ($arg(\widehat{g})$)



(b) Widmo amplitudowe ($|\widehat{g}|$)

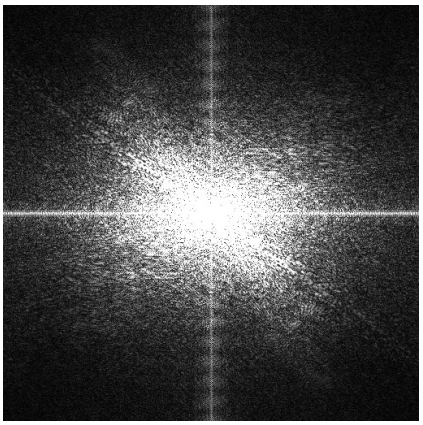
Rysunek 12: Widma obrazka Lena.bmp (niskie częstotliwości w centrum)

Transformata Fouriera jest transformatą globalną, to oznacza, że zmiana w stosunkowo niewielkim obszarze, zmienia całą transformatę, co widać na poniższym przykładzie:

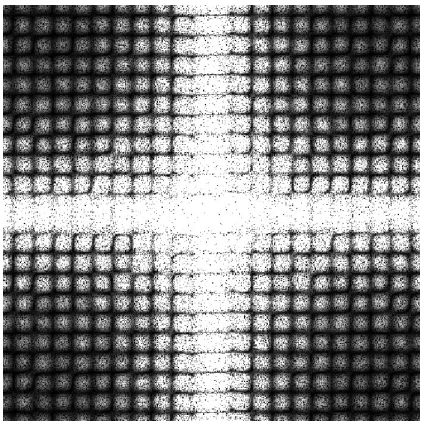
Rysunek 13: Zmodyfikowany obrazek g' Lena.bmp

```
clc;
clear;
[A,map]=imread('Lena.bmp','bmp');

for i=300:320
    for j=300:320
        A(i,j)=0;
    end
end

b=fft2(A);
c = fftshift( abs(b) );
d = fftshift( (pi + angle(b))*128/pi );

figure(1);
colormap(map);
image( A );
title('Zmodyfikowany obrazek:');
axis off;
axis equal;
```

```
figure(2);
colormap(map);
image( c/50 );
title('Widmo fazowe')
axis off;
axis equal;

figure(3);
colormap(map);
image( d );
title('Widmo amplitudowe')
axis off;
axis equal;
```



(a) Widmo amplitudowe ($|\widehat{g'}|$)



(b) Moduł różnicy ($|\widehat{g'} - \widehat{g}|$)

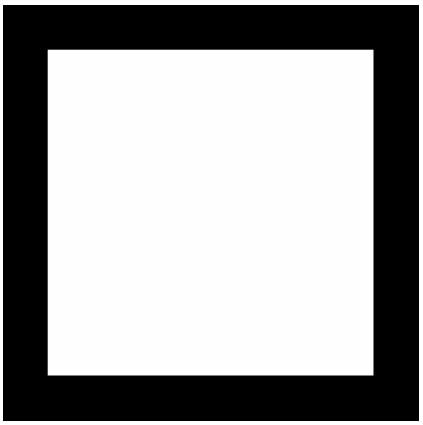
Rysunek 14: Widma obrazków Lena.bmp (niskie częstotliwości w centrum)

Sprawdźmy teraz jakie skutki może mieć kompresja. W tym celu nałożymy maskę filtru dolnoprzepustowego na transformatę obrazka, i spróbujemy odtworzyć obrazek.

```
clc;
clear;
[A,map]=imread('Lena.bmp','bmp');
A=double(A);
maska=zeros(512);

bok=400;
maska=zeros(512);
for i=(512 - bok)/2:(512 + bok)/2
    for j=(512 - bok)/2:(512 + bok)/2
        maska(i,j)=1;
    end
end

B=fftshift(fft2(A));
B=maska.*B;
B=ifft2(ifftshift(B));
B=uint8(abs(B));

figure(1);
colormap(map);
image(uint8(A));
title('Oryginal');
axis off;
axis equal;

figure(2);
colormap(map);
image( maska*255  );
title('Maska');
axis off;
axis equal;

figure(3);
colormap(map);
image(B);
```

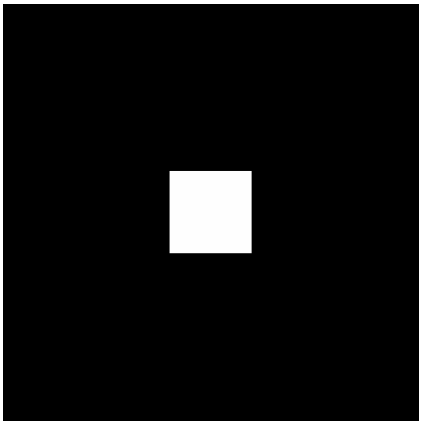
```
37 title('Odtworzony obrazek');
38 axis off;
39 axis equal;
```

Rysunek 15: Oryginał



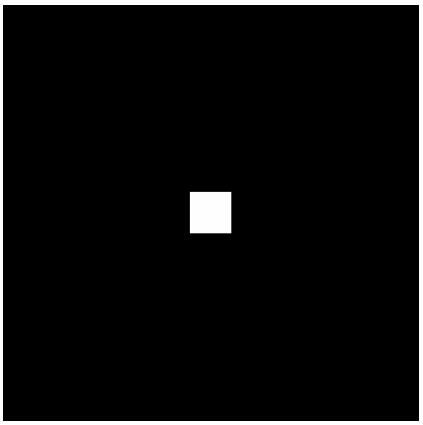
(a) Maska bok=400

(b) Po przefiltrowaniu bok=400



(c) Maska bok=100

(d) Po przefiltrowaniu bok=100



(e) Maska bok=50

(f) Po przefiltrowaniu bok=50

Rysunek 16: Skutki kompresji dla różnych masek

W pobliżu krawędzi można zaobserwować tzw. efekt Gibbsa. Zjawisko to powstaje tam, gdzie sygnał jest aproksymowany szeregami Fouriera. Wówczas przy krawędziach (czyli w miejscach gdzie osiągane są wysokie częstotliwości) obserwuje się skok i oscylacje. Widać to doskonale przy próbie aproksymacji fali prostokątnej. W przypadku obrazów efekt Gibbsa objawia się tym, że w pobliżu krawędzi powstają cienie przypominające fale na wodzie. Prześledźmy jego powstawanie w zależności od maski:

(a) Oryginał

(b) Maska bok=400

(c) Maska bok=300

(d) Maska bok=250

Rysunek 17: Efekt Gibbsa maski: 400, 300, 250

(a) Maska bok=200

(b) Maska bok=150

(c) Maska bok=100

(d) Maska bok=50

Rysunek 18: Efekt Gibbsa maski: 200, 150, 100, 50

Minimalizację efektu Gibbsa osiąga się poprzez minimalizowanie zjawiska przecieków widma związanych z próbkowaniem sygnału, gdy w widmie oryginalnego sygnału istnieją składowe o pulsacjach różnych od pulsacji współczynników transformaty (czyli różnych od pulsacji $\frac{2\pi n}{N}$). Wówczas dochodzi do przecieków, które wpływają na wszystkie wartości pozostałych współczynników. Aby zredukować to zjawisko, dzieli się obraz na bloki i aplikuje transformaty do każdego bloku z osobna. W praktyce najczęściej używane są bloki od 4x4 do 32x32. Wielkość bloku jest powiązana ze szczegółowością zdjęcia: jeśli ma ono dużo krawędzi, wielkości bloków się zwiększa. Przyjmuje się, że bloki 8x8 są optymalne w obydwu przypadkach.

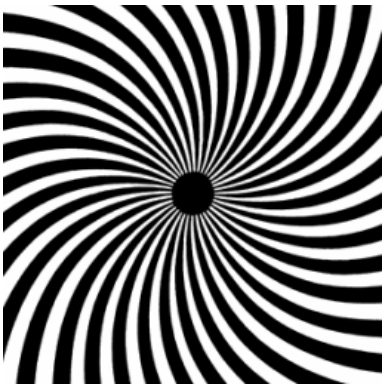
Prześledźmy teraz efekt Gibbsa na zdjęciach o różnym stopniu szczegółowości

(a) Oryginał

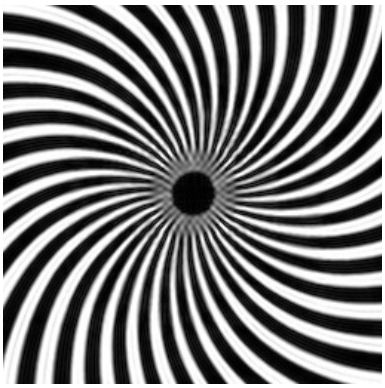
(b) Przefiltrowany

(c) Oryginał

(d) Przefiltrowany



(e) Oryginał



(f) Przefiltrowany

Rysunek 19: Kilka przykładów kompresji dla maski bok=200 (dobrana tak, aby zapewnić dobrą jakość obrazka nr 1). Obserwuje się większe zakłócenia na pozostałych obrazach zaś największe są na dolnym obrazku.

4.3 Transformata kosinusowa

Motywacją do użycia transformaty kosinusowej, zamiast Fouriera, jest fakt, że pozbędziemy się problemu prowadzenia obliczeń na liczbach zespolonych. Po przeprowadzeniu wielu badań wywnioskowano, że transformata kosinusowa stanowi najlepsze przybliżenie optymalnej transformaty KLT spośród wszystkich transformacji unitarnych o szybkich algorytmach obliczeniowych dla sygnałów losowych o dużym współczynniku korelacji [4, s. 9]. Najlepiej spośród badanych transformat nadaje się do przetwarzania fotografii, bo bardzo dobrze „upakowuje" energię obrazu we współczynnikach odpowiadających niskim częstotliwościom [1, s. 353]. Inna sprawa to fakt, że wzór na DCT jest bardzo prosty, dzięki czemu w praktycznych realizacjach nie powoduje ona dodatkowych problemów.

Zacznijmy od transformaty Fouriera w wersji ciągłej:

$$\widehat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\ e^{-2\pi i x\xi}\ dx = \int_{-\infty}^{\infty} f(x)\left(\cos\left(-2\pi x\xi\right) + i\ \cdot \sin\left(-2\pi x\xi\right)\right)\ dx =$$

$$= \int_{-\infty}^{\infty} f(x)\ \cos\left(-2\pi x\xi\right)\ dx + i\ \int_{-\infty}^{\infty} f(x)\ \sin\left(-2\pi x\xi\right)\ dx = (*)$$

Zakładając, że f jest funkcją parzystą (tzn. $f(x) = f(-x)$) mamy:

$$\int_{-\infty}^{\infty} f(x)\ \sin\left(-2\pi x\xi\right)\ dx = -\int_{0}^{\infty} f(x)\underbrace{\left(\sin\left(2\pi x\xi\right) + \sin\left(-2\pi x\xi\right)\right)}_{0}\ dx = 0$$

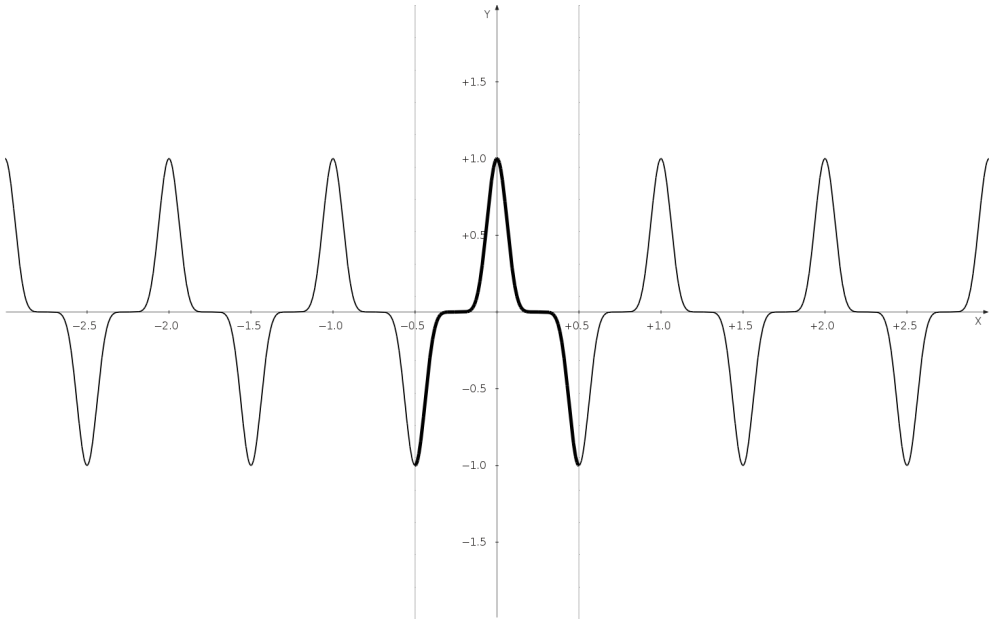
Podstawiając, otrzymujemy wzór na ciągłą transformatę kosinusową:[16].

$$\widehat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\ \cos\left(-2\pi x\xi\right)\ dx$$

[16]Warto zauważyć, że jeśli f jest funkcją rzeczywistą to transformata kosinusowa też jest rzeczywista, i dodatkowo parzysta dzięki czemu wzór na przekształcenie odwrotne jest zbliżony do tego na transformatę.

W praktyce nie będziemy przetwarzać sygnałów ciągłych, tylko dyskretne. W związku z tym koniecznym jest wyprowadzenie dyskretnej transformaty.

Niech f - funkcja parzysta o okresie 1.



Obcinając tę funkcję do przedziału $\left[-\frac{1}{2}, \frac{1}{2}\right]$ oraz biorąc:

$$c_n = \int_{-\frac{1}{2}}^{\frac{1}{2}} f(x)\ \cos\left(-2\pi x\xi\right)\ dx$$

gdzie $c_n = c_{-n}$ oraz c_n jest rzeczywiste, mamy:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{2i\pi nx} = c_0 + \sum_{n=1}^{\infty} c_n \left(e^{2i\pi nx} + e^{-2i\pi nx}\right) = c_0 + 2\sum_{n=1}^{\infty} c_n \cos(2i\pi nx)$$

Podstawiając $f(x) := \sum_k a_k \delta_{(2k+1)h}$ z krokiem $h = \frac{1}{2N}$ mamy:

$$c_n = \sum_k a_k\ \cos\left(\frac{2\pi(2k+1)n}{2N}\right), \qquad c_{n+N} = -c_n$$

Następnie licząc wartość średnią wyrażenia $c_0 + 2\sum_{n=1}^{m} c_n \ \cos(2\pi nx)$:

$$\lim_{M\to\infty} \frac{1}{M} \sum_{m=M}^{2M} \left(c_0 + 2\sum_{n=1}^{m} c_n \ \cos(2\pi nx) \right) = \sum_{k} \left(\sum_{n=0}^{N-1} c_n \ \cos\left(\frac{2\pi nk}{N}\right) \right) \delta_{\frac{k}{N}}$$

Zatem transformatą kosinusową sygnału jednowymiarowego będziemy nazywać sumę: $\sum_{n=0}^{N-1} c_n \ \cos\left(\frac{2\pi nk}{N}\right)$.

W celu uproszczenia w dalszej części pracy transformatę kosinusową będziemy określać wzorem:

$$\widehat{f}(k) = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} f(n) \ \cos\left(\frac{\pi(2n+1)k}{2N}\right)$$

Zaś w praktyce DCT będzie aplikowana do bloków rozmiaru 8x8, wówczas wzór jest następujący:

$$\widehat{f}(k) = \frac{1}{2} C(k) \sum_{n=0}^{7} f(n) \ \cos\left(\frac{\pi(2n+1)k}{2N}\right) \tag{10}$$

gdzie

$$C(a) = \begin{cases} \frac{1}{\sqrt{2}} & \text{dla a} = 0 \\ 1 & \text{dla a} \neq 0 \end{cases}$$

Dowód:

Weźmy transformatę:

$$\widehat{f}(k) = \sqrt{\frac{2}{N}} \ C(k) \sum_{n=0}^{N-1} f(n) \ \cos\left(\frac{\pi(2n+1)k}{2N}\right)$$

Pokażemy, że wzór:

$$f(n) = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} C(k) \ \widehat{f}(k) \ \cos\left(\frac{\pi(2n+1)k}{2N}\right)$$

definiuje transformatę odwrotną, dla C(k) jak we wzorze (10).

Aby kontynuować wprowadzimy sobie prostą tożsamość przy pomocy wzoru Eulera: $\cos x = \frac{e^{ix}+e^{-ix}}{2}$

$$\sum_{n=0}^{N-1} \cos\left[\frac{\pi(2l+1)k}{2N}\right] \cdot \cos\left[\frac{\pi(2n+1)k}{2N}\right] =$$

$$= \frac{1}{2}\sum_{n=0}^{N-1} \cos\left[\frac{2\pi k(l+n+1)}{2N}\right] + \cos\left[\frac{2\pi k(l-n)}{2N}\right]$$

Zacznijmy od wzoru na transformatę odwrotną:

$$f(n) = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} C(k)\ \widehat{f}(k)\ \cos\left(\frac{\pi(2n+1)k}{2N}\right)$$

Wstawiając do niej wzór na transformatę, otrzymujemy:

$$f(n) = \frac{2}{N} \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} [C(k)]^2 f(l)\ \cos\left[\frac{\pi(2l+1)k}{2N}\right] \cos\left[\frac{\pi(2n+1)k}{2N}\right] \ =$$

Po podstawieniu tożsamości:

$$= \frac{1}{N} \sum_{l=0}^{N-1} f(l) \sum_{k=0}^{N-1} [C(k)]^2 \left(\cos\left[\frac{2\pi k(l+n+1)}{2N}\right] + \cos\left[\frac{2\pi k(l-n)}{2N}\right]\right)$$

Zajmiemy się teraz sumą:

$$\sum_{k=0}^{N-1} [C(k)]^2 \left(\cos\left[\frac{2\pi k(l+n+1)}{2N}\right] + \cos\left[\frac{2\pi k(l-n)}{2N}\right]\right) \ =$$

$$= \left[\sum_{k=0}^{N-1} [C(k)]^2 \cos\left[\frac{2\pi k(l+n+1)}{2N}\right]\right] + \left[\sum_{k=0}^{N-1} [C(k)]^2 \cos\left[\frac{2\pi k(l-n)}{2N}\right]\right] \ =$$

$$= \left[\sum_{k=0}^{N-1} [C(k)]^2 \cos\left[k\frac{2\pi(l+n+1)}{2N}\right]\right] + \left[\sum_{k=0}^{N-1} [C(k)]^2 \cos\left[k\frac{2\pi(l-n)}{2N}\right]\right] \ =$$

Dla $a = \frac{2\pi(l+n+1)}{2N}$ oraz b=$\frac{2\pi(l-n)}{2N}$ mamy:

$$= \left[\sum_{k=0}^{N-1} [C(k)]^2 \cos[k \cdot a]\right] + \left[\sum_{k=0}^{N-1} [C(k)]^2 \cos[k \cdot b]\right] =$$

Stosując teraz wzór na postać wykładniczą liczby zespolonej dostajemy:

$$= \left[\sum_{k=0}^{N-1} [C(k)]^2 e^{kai}\right] + \left[\sum_{k=0}^{N-1} [C(k)]^2 e^{kbi}\right] =$$

Pamiętać tylko trzeba, żeby potem brać część rzeczywistą. Zapisując nieco inaczej:

$$= \left[\sum_{k=0}^{N-1} [C(k)]^2 \left(e^{ai}\right)^k\right] + \left[\sum_{k=0}^{N-1} [C(k)]^2 \left(e^{bi}\right)^k\right] =$$

widać, że dostajemy szeregi prawie geometryczne. Sprawę psuje tylko element C(k), którego łatwo się pozbędziemy, bo dla $k > 0$ jest równy 1 :

$$= \left[[C(0)]^2 \left(e^{ai}\right)^0 + \sum_{k=1}^{N-1} \left(e^{ai}\right)^k\right] + \left[[C(0)]^2 \left(e^{bi}\right)^0 + \sum_{k=1}^{N-1} \left(e^{bi}\right)^k\right] =$$

$$= \left[\frac{1}{2} + \sum_{k=1}^{N-1} \left(e^{ai}\right)^k\right] + \left[\frac{1}{2} + \sum_{k=1}^{N-1} \left(e^{bi}\right)^k\right] =$$

Przesuwając nieco granice sumowania dostajemy:

$$= \left[-\frac{1}{2} + \sum_{k=0}^{N-1} \left(e^{ai}\right)^k\right] + \left[-\frac{1}{2} + \sum_{k=0}^{N-1} \left(e^{bi}\right)^k\right] =$$

$$= \left[\sum_{k=0}^{N-1} \left(e^{ai}\right)^k\right] + \left[\sum_{k=0}^{N-1} \left(e^{bi}\right)^k\right] - 1 =$$

Teraz możemy wysumować obydwa szeregi geometryczne:

$$= \frac{1-(e^{ai})^N}{1-e^{ai}} + \frac{1-\left(e^{bi}\right)^N}{1-e^{bi}} - 1 =$$

$$= \frac{1-e^{aiN}}{1-e^{ai}} + \frac{1-e^{biN}}{1-e^{bi}} - 1, \qquad a = \frac{2\pi(l+n+1)}{2N},\ b = \frac{2\pi(l-n)}{2N}$$

Zastanówmy się teraz chwilkę, co dostaliśmy. Dla $a = \frac{2\pi(l+n+1)}{2N}$ oraz $l <= N-1$ i $n <= N-1$, czyli $l+n+1 <= 2N-1$ dostajemy, że $0 < a < 2\pi$. Stąd $1-e^{ai}$ nie jest 0. Jeśli $k+l+1$ jest parzyste, to dostajemy aN, które jest wielokrotnością liczby 2π. Stąd e^{aiN} jest równy 1, czyli $1 - e^{aiN} = 0$ zatem $\frac{1-e^{aiN}}{1-e^{ai}} = 0$.

Pozostaje sprawdzić przypadek, gdy $k + l + 1$ jest nieparzyste, wówczas $k + l + 1 = 2s + 1$ dla pewnego całkowitego s. Stąd

$$e^{aiN} = e^{i\pi(2s+1)} = \cos(\pi(2s+1)) + i\sin(\pi(2s+1))$$

Dla każdego całkowitego s, $\cos(\pi(2s+1)) = -1$, a $\sin(\pi(2s+1)) = 0$. Stąd $e^{aiN} = -1$, oraz $1 - e^{aiN} = 2$. Zatem mamy:

$$\frac{2}{1-e^{ai}}$$

Aby policzyć część rzeczywistą ułamka zastosujemy uproszczenia:

$$\frac{2}{1-e^{ai}} = \frac{2(1-e^{-ai})}{(1-e^{ai})(1-e^{-ai})} = \frac{2(1-e^{-ai})}{-e^{ai}-e^{-ai}+2} =$$

$$= \frac{2(1-e^{-ai})}{-(\cos(a)-i\sin(a)) - (\cos(-a)-i\sin(-a)) + 2} =$$

$$= \frac{2(1-e^{-ai})}{-\cos(a)+i\sin(a)-\cos(a)-i\sin(a)+2} =$$

$$= \frac{2(1-e^{-ai})}{2-2\cos(a)} = \frac{2(1-e^{-ai})}{2(1-\mathrm{Re}(e^{ai}))}$$

Teraz nakładając część rzeczywistą na ułamek:

$$\mathrm{Re}\left(\frac{2}{1-e^{ai}}\right) = \mathrm{Re}\left(\frac{2(1-e^{-ai})}{2(1-\mathrm{Re}(e^{ai}))}\right)$$

Ponieważ mianownik jest rzeczywisty, wchodzimy z częścią rzeczywistą do licznika otrzymując:

$$\frac{2(1-\mathrm{Re}(e^{-ai}))}{2(1-\mathrm{Re}(e^{ai}))} = \frac{2(1-\mathrm{Re}(e^{ai}))}{2(1-\mathrm{Re}(e^{ai}))} = 1$$

W podobny sposób rozważmy $b = \frac{2\pi(l-n)}{2N}$, z tą różnicą, że b jest zerem dla $n = l$

Dla $l - n$ parzystych, oraz $l - n$ nieparzystych, sposób rozumowania jest identyczny jak w przypadku dla a. Pozostaje sprawdzić, co się dzieje $l-n = 0$. Wówczas $b = 0$, i dostajemy wyrażenie nieoznaczone $\frac{1-e^{biN}}{1-e^{bi}} = \frac{0}{0}$. Cofnijmy się zatem do postaci tego wyrażenia przed wysumowaniem:

$$\sum_{k=0}^{N-1} \left(e^{bi}\right)^k$$

Dla $b = 0$ dostajemy:

$$\sum_{k=0}^{N-1} \left(e^{0}\right)^k = \sum_{k=0}^{N-1} 1 = N$$

Podsumowując powyższe:

	$l+n+1$	$l-n$
Parzyste	0	0
Nieparzyste	1	1
0	brak	N

Weźmy najpierw $l-n = 2s,\ s \in \mathbb{Z}$. Wtedy $l = 2s+n$, i sprawdźmy parzystość drugiego wyrażenia: $l+n+1 = 2s+2n+1$. Czyli dla $l-n$ parzystego, $l+n+1$ jest nieparzyste.

Weźmy teraz $l - n = 2s + 1,\ s \in \mathbb{Z}$. Wtedy $l = 2s + n + 1$, podstawiając otrzymujemy: $l + n + 1 = 2s + 2n + 2$, czyli parzyste.

Weźmy teraz $l - n = 0$. Wtedy $l = n$, i po podstawieniu otrzymujemy: $l + n + 1 = 2l + 1$, czyli nieparzyste.

Spójrzmy teraz całościowo na wyrażenie:

$$\frac{1-e^{aiN}}{1-e^{ai}}+\frac{1-e^{biN}}{1-e^{bi}}-1$$

Ponieważ l i n ustalone, to możliwe są dwie jego wartości:

$$\begin{cases} N & \text{dla } \mathrm{l}=\mathrm{n} \\ 0 & \text{dla } \mathrm{l}\neq\mathrm{n} \end{cases}$$

Układ jest ortogonalny, więc elementy są dokładnie odtworzone

4.3.1 Dwuwymiarowa dyskretna transformata kosinusowa

Dwuwymiarową dyskretną transformatą kosinusową dla bloków 8x8 pikseli będziemy nazywać przekształcenie dane wzorem:

$$\widehat{f}(x,y) = \frac{1}{4}\sum_{n=0}^{7}\sum_{k=0}^{7} f(n,k)\cdot\cos\left(\frac{(2n+1)x\pi}{16}\right)\cdot\cos\left(\frac{(2k+1)y\pi}{16}\right) \qquad (11)$$

gdzie

$$C(a) = \begin{cases} \frac{1}{\sqrt{2}} & \text{dla a} = 0 \\ 1 & \text{dla a} \neq 0 \end{cases}$$

W celu zmniejszenia ilości obliczeń, analogicznie do zwykłej transformaty Fouriera, korzysta się z własności separowalności oraz odpowiednika FFT dla DCT.

4.3.2 Odwrotna dwuwymiarowa transformata kosinusowa

Odwrotną dwuwymiarową dyskretną transformatą kosinusową dla bloków 8x8 pikseli będziemy nazywać przekształcenie dane wzorem:

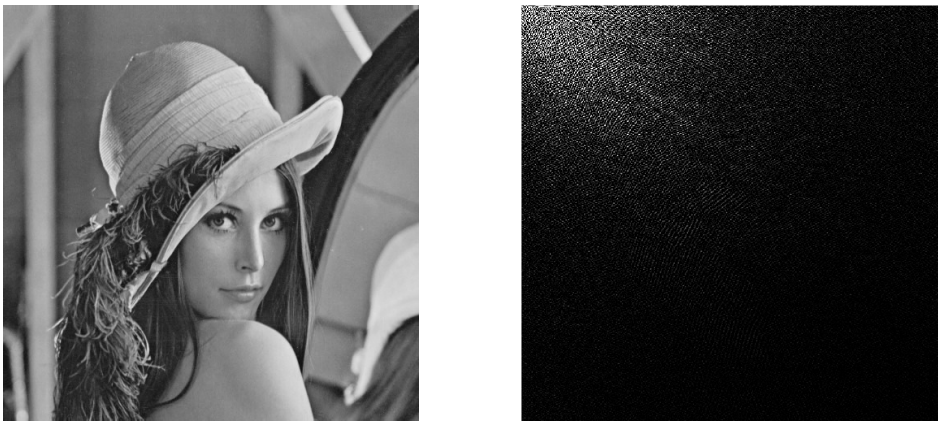
$$f(x,y) = \frac{1}{4}\sum_{n=0}^{7}\sum_{k=0}^{7} \widehat{f}(n,k)\cdot\cos\left(\frac{(2x+1)n\pi}{16}\right)\cdot\cos\left(\frac{(2y+1)k\pi}{16}\right) \qquad (12)$$

gdzie

$$C(a) = \begin{cases} \frac{1}{\sqrt{2}} & \text{dla a} = 0 \\ 1 & \text{dla a} \neq 0 \end{cases}$$

4.3.3 Działanie transformaty kosinusowej

Sprawdźmy w praktyce działanie transformaty kosinusowej na obrazku Lena. Na początek policzymy widmo obrazka - w tym celu użyjemy programu Matlab, oraz jego wbudowanej funkcji: **dct2** (dwuwymiarowa dyskretna transformata kosinusowa). Zobaczymy, że niskie częstotliwości w widmie będą w lewym górnym rogu, a najwyższe w prawym dolnym.



(a) Lena.bmp (b) Widmo

Rysunek 21: Obrazek Lena.bmp oraz jego widmo.

Sprawdźmy teraz skutki użycia DCT. W tym celu użyjemy skryptu:

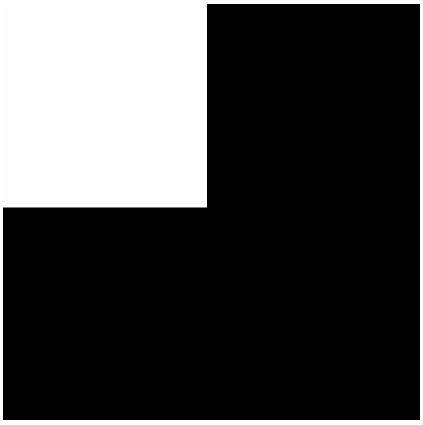
```
clc;
clear;
[A,map]=imread('Lena.bmp','bmp');
b=dct2(A);
mask=zeros(512);
bok=50;

for i=1:bok
    for j=1:bok
        mask(i,j)=1;
    end
end
b=b.*mask;
c=idct2(b);

figure (1)
image(A)
colormap(map)
title('Oryginalny obrazek');
axis off
axis equal

figure (2)
image(c)
colormap(map)
title('Odtworzony obrazek');
axis off
axis equal
```

(a) Maska bok=50

(b) Po przefiltrowaniu bok=50

(c) Maska bok=100

(d) Po przefiltrowaniu bok=100



(e) Maska bok=250

(f) Po przefiltrowaniu bok=250

Rysunek 22: Skutki kompresji dla różnych masek - widoczny efekt Gibbsa.

4.3.4 Minimalizacja efektu Gibbsa

Jak zauważyliśmy w poprzednim podrozdziale, transformata kosinusowa nie jest wolna od efektu Gibbsa. W podrozdziale 4.2.1 wspomnieliśmy, że zjawisko to minimalizuje się dzieląc obraz na bloki, i przeprowadzając kompresję każdego bloku z osobna. Sprawdźmy w praktyce działanie tej metody.

Za przykładowy obrazek posłuży nam zdjęcie Lena. Zakładamy, że wielkości bloków to 8x8 pikseli. W celu kompresji użyjemy poniższych masek:

1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabela 1: Maska nr 0.

1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabela 2: Maska nr 1.

1	1	1	0	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabela 3: Maska nr 2.

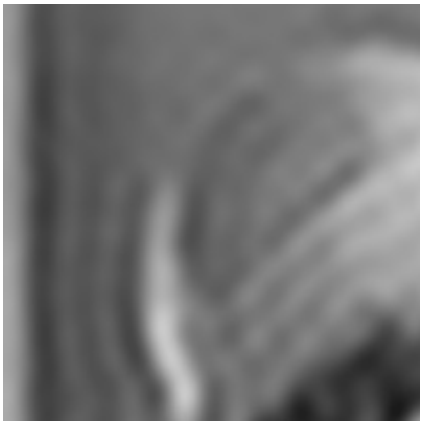
Skonfrontujemy to z użyciem maski bok=64, bok=128 i bok=192 odpowiednio dla masek nr 0, 1 i 2. Wartości 64, 128 i 192 wzięte są nieprzypadkowo: są proporcjonalne do użytych masek rozmiaru 8x8 ($\frac{3}{8} \cdot 512 = 192$).

```
clc;
clear;

[A,map]=imread('Lena.bmp', 'bmp');

AA=zeros(512);
B=zeros(8);

maska0=zeros(8);
      maska0(1,1)=1;

maska1=zeros(8);
      maska1(1,1)=1;
      maska1(1,2)=1;
      maska1(2,1)=1;
      maska1(2,2)=1;

maska2=zeros(8);
      maska2(1,1)=1;
      maska2(1,2)=1;
      maska2(1,3)=1;
      maska2(2,1)=1;
      maska2(2,2)=1;
      maska2(2,3)=1;
      maska2(3,1)=1;
      maska2(3,2)=1;
      maska2(3,3)=1;


for i=1:8:512
      for j=1:8:512

          %wczytanie bloku
               for k=0:7
                    for l=0:7
                         B(k+1,l+1)=A(i+k,j+l);
                    end
               end
```

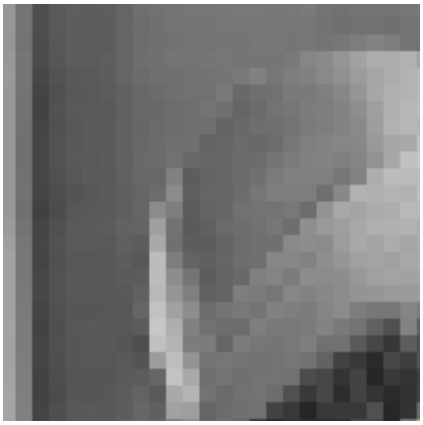
```
41          %transformata
42              C=dct2(B);
43          %maska
44              C=C.*maska2;
45          %odwrotna transformata
46              D=idct2(C);
47          %przepisanie bloku do~obrazka
48              for k=0:7
49                  for l=0:7
50                      AA(i+k,j+l)=D(k+1,l+1);
51                  end
52              end
53      end
54  end


57  figure (1)
58  image(AA)
59  colormap(map)
60  axis off
61  axis equal
```

(a) Zwykłe DCT, bok=64

(b) DCT blok 8x8, maska nr 0



(c) Zwykłe DCT, bok=64

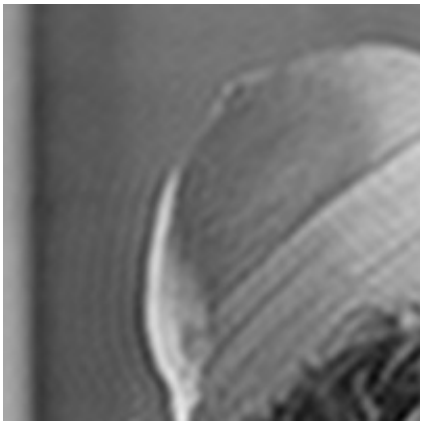


(d) DCT blok 8x8, maska nr 0

Rysunek 23: Odtworzenie rysunku na podstawie składnika stałego, i proporcjonalnie zwykła transformata DCT.

(a) Zwykłe DCT, bok=128

(b) DCT blok 8x8, maska nr 1



(c) Zwykłe DCT, bok=128

(d) DCT blok 8x8, maska nr 1

Rysunek 24: Odtworzenie rysunku na podstawie składnika stałego + trzech składników zmiennych, i proporcjonalnie zwykła transformata DCT.

(a) Zwykłe DCT, bok=192

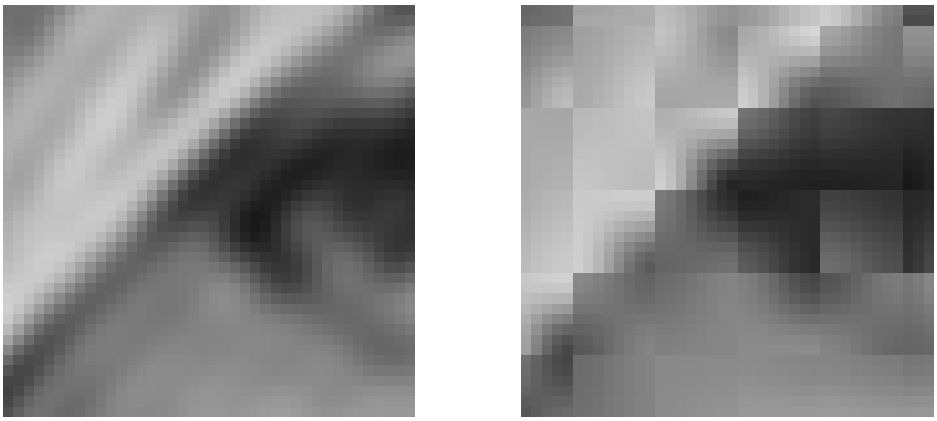
(b) DCT blok 8x8, maska nr 2

(c) Zwykłe DCT, bok=192

(d) DCT blok 8x8, maska nr 2

Rysunek 25: Odtworzenie rysunku na podstawie składnika stałego + ośmiu składników zmiennych, i proporcjonalnie zwykła transformata DCT.

Co prawda pozbyliśmy się efektu Gibbsa, ale odbyło się to kosztem pojawienia się efektu blokowego, jeszcze lepiej widocznego po dalszym powiększeniu:



(a) Zwykłe DCT, bok=128 (b) DCT blok 8x8, maska nr 1

Rysunek 26: Efekt blokowy w przypadku blokowej DCT.

Czytelnik może odnieść wrażenie, że zamiana DCT na blokową DCT przy takich efektach, i określanie tego mianem poprawienia jakości, to nieporozumienie, i realizacja przysłowia „zamienił stryjek siekierkę na kijek". Oczywiście maski w przykładzie dobrane zostały tak, aby przy swojej prostocie uwidocznić pewne cechy transformaty i wywołać efekt Gibbsa. W praktyce takich filtrów się nie stosuje, bo mają fatalne właściwości, o czym się przekonaliśmy powyżej. W praktycznych realizacjach filtry dobiera się tak, aby przy jednoczesnej eliminacji efektu Gibbsa, efekt blokowy był znikomy.

4.4 Stratne kodowanie transformatowe: ogólna idea

Przed przystąpieniem do kompresji, na początek należy przekonwertować sygnał do przestrzeni YCbCr (lub innej, którą uznamy za lepszą). W dalszych krokach każda z tych składowych będzie przetwarzana oddzielnie.

4.4.1 Wyznaczenie widma

Do wyznaczenia widma użyjemy dyskretnej transformaty kosinusowej. Ma to na celu zdekorelowanie wartości próbek i tym samym pozbycie się redundancji w sygnale. Każdą składową sygnału dzielimy na bloki rozmiaru 8x8 i aplikujemy do każdego z osobna DCT dane wzorem (11) (rozdz. 4.3.1).

W efekcie dla każdego bloku 8x8 otrzymujemy tablicę współczynników transformaty rozmiaru 8x8 o wartościach rzeczywistych, czyli widmo bloku.

$\widehat{f}(0,0)$ nazywamy składnikiem stałym, choć w literaturze bywa nazywany współczynnikiem stałoprądowym, lub współczynnikiem DC. Pozostałe nazywamy składnikami zmiennymi, choć bywają nazywane współczynnikami zmiennoprądowymi lub współczynnikami AC. Składnik stały ma tę własność, że ma większą wartość, niż składniki zmienne, które na ogół są bliskie 0. Wiedząc to, możemy tak przeprowadzić kwantyzację, aby uzyskać dobry współczynnik kompresji kosztem niewielkiego pogorszenia jakości odtworzonego sygnału.

4.4.2 Kwantyzacja próbek widma

Kolejnym etapem jest kwantyzacja współczynników. Różne algorytmy stosują różne metody. Ogólnie zasada kwantyzacji polega na tym, aby z większą dokładnością kodować składowe niskich częstotliwości. Dzieje się tak dlatego, że wzrok ludzki jest bardziej czuły na niższe częstotliwości przestrzenne.

4.4.3 Kodowanie bezstratne

Ostatnim etapem jest kodowanie entropijne: składniki zmienne koduje się osobno, składniki stałe osobno. Na początku współczynniki w każdym bloku

porządkuje się metodą zygzakowatą:

1	2	6	7	15	16	28	29
3	5	8	14	17	27	30	43
4	9	13	18	26	31	42	44
10	12	19	25	32	41	45	54
11	20	24	33	40	46	53	55
21	23	34	39	47	52	56	61
22	35	38	48	51	57	60	62
36	37	49	50	58	59	63	64

Tabela 4: Kolejność pól w porządku zygzakowatym

Ponieważ składniki stałe mają małe wartości i są silnie kwantyzowane, toteż sporo z nich jest równe 0. Dlatego kodowaniu poddaje się parę (r, l), gdzie r to liczba zerowych współczynników poprzedzających niezerowy składnik zmienny, zaś l to wartość owego niezerowego składnika. Takie pary koduje się z użyciem kodów Huffmana lub kodami arytmetycznymi.

Jeśli chodzi o składniki stałe: stanowią one informację o średniej wartości w bloku. Na dodatek są silnie między sobą skorelowane. W związku z tym stosuje się predykcję DPCM, zaś jako predyktora używa się poprzedniego współczynnika DC. Dzięki temu jesteśmy w stanie wyliczyć błąd predykcji, który stanowi dla nas informację, i jest kodowany kodami arytmetycznymi lub Huffmana.

5 Stratne kodowanie transformatowe: JPEG

JPEG (Joint Photographic Experts Group) to międzynarodowy standard kompresji statycznych obrazów cyfrowych o ciągłej skali odcieni (digital compression and coding of continuous-tone still images).

5.1 Podstawy

Standard obejmuje obrazy zarówno kolorowe jak i czarno-białe, przy czym obrazy kolorowe traktuje się jako obrazy złożone z kilku składowych obrazów - w zależności od użytej przestrzeni barw. Kodowaniu poddaje się każdy obraz składowy z osobna. Algorytm JPEG może pracować na każdej ciągłej przestrzeni barw, i tę własność nazywa się: colorblindness of JPEG.

Standard składa się z dwóch części: Requirements and guidelines oraz Compilance testing. W dalszej części pracy opisana zostanie zawartość pierwszej części. Druga dotyczy testów weryfikujących konkretną implementację ze standardem.

5.1.1 Tryby pracy

W standardzie JPEG używa się czterech trybów pracy [10, s. 17]:

1. **Sekwencyjny oparty o DCT** - typowe bloki wejściowe rozmiaru 8x8, czytane od lewej do prawej, wiersz po wierszu, począwszy od góry. Po wyznaczeniu widma, jego skwantyzowaniu i przygotowaniu do kodowania entropijnego, wszystkie 64 współczynniki transformaty mogą być poddane tejże kompresji od razu, a następnie zapisane na wyjściu jako skompresowany sygnał.

2. **Progresywny oparty o DCT** - bloki 8x8 czytane jak wyżej, z tą różnicą że robi się to w wielu przebiegach. W tym celu umieszcza się bufor między kwantyzatorem, a koderem entropijnym - każdy blok po transformacji i kwantyzacji jest w nim zapisywany. Współczynniki transfor-

maty zapisane w owym buforze są następnie kodowane w wielu przebiegach.

3. **Bezstratny** - bez udziału DCT i kwantyzatora.

4. **Hierarchiczny** - obraz kodowany jest jako ciąg ramek, w wielu rozdzielczościach, dzięki czemu odtworzenie obrazu do małej rozdzielczości wymaga przeprowadzenia mniejszej liczby obliczeń, niż odtwarzanie pełnej rozdzielczości.

Każdy z trybów zostanie bardziej szczegółowo opisany w dalszej części pracy, ze szczególnym uwzględnieniem trybu sekwencyjnego, który jest podstawowym trybem pracy JPEG'a.

5.1.2 Kodowanie entropijne

Dostępne są dwie metody kodowania entropijnego: kodami Huffmana lub kodami arytmetycznymi. Nie specyfikuje się domyślnych zawartości tablic kodów Huffmana, w przeciwieństwie do kodów arytmetycznych, dla których są zdefiniowane domyślne wartości tablic [10, s. 18]. Częściej stosuje się kody Huffmana z uwagi na to, że kody arytmetyczne są objęte ochroną patentową w USA.

5.1.3 Dokładności próbek

Dla kodowania z wykorzystaniem DCT dostępne są dwa stopnie precyzji próbki: 8-bitowy i 12-bitowy. Domyślnie używa się 8-bitowych próbek (obliczenia na 12-bitowych próbkach są bardziej kosztowne, zarówno jeśli chodzi o czas procesora, jak i pamięć). Natomiast dla kodowania bezstratnego dostępne precyzje to: 2-, 3-, ..., 16-bitów.

5.2 Tryb sekwencyjny

Jest to tryb najczęściej wykorzystywany. Jak już wspomnieliśmy w podrozdziale 4.1, tryb składa się z trzech części, które kolejno omówimy.

Na początku (jeśli to obraz kolorowy) wykonujemy transformację przestrzeni barw do YCbCr. W dalszej kolejności będziemy przetwarzać każdą składową z osobna.

5.2.1 Wyznaczenie widma

Obraz dzielimy na bloki 8x8 pikseli, i dla każdego bloku aplikujemy dyskretną transformatę kosinusową daną wzorem (11). W efekcie uzyskujemy składnik stały i 63 składniki zmienne. Sygnał jest przygotowany do drugiego etapu.

5.2.2 Kwantyzacja próbek widma

Współczynniki DCT poddajemy kwantyzacji równomiernej zgodnie ze wzorem:

$$\widehat{f}^{Q}(x,y) = round\left(\frac{\widehat{f}(x,y)}{Q(x,y)}\right) \tag{13}$$

gdzie Q to tabela kwantyzacji (wartości od 1 do 255) zdefiniowana w kwantyzatorze. Stosuje się oddzielne tabele kwantyzacji dla luminancji i chrominancji. Ponieważ wzrok ludzki lepiej wyłapuje różnice w poziomach jasności, niż w kolorach, toteż chrominancję będziemy kwantyzować stosując mniej wartości (większe przedziały). Od zastosowanych tabel zależy jakość obrazu i stopień kompresji. Poniżej są przykładowe tabele kwantyzacji dla poszczególnych składowych (źródło: [10, Annex K]).

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Tabela 5: Tabela kwantyzacji składowej luminancji

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Tabela 6: Tabela kwantyzacji składowej chrominancji.

Chcąc zwiększyć przedziały kwantyzacji (i tym samym kompresję) należy zwiększyć współczynniki tablic kwantyzacji. Porównując to z przykładem

w podrozdziale 4.3.4 widać, jak łagodne filtry się stosuje. We wspomnianym podrozdziale ustawiono bardzo „twardy" filtr, którego pasmo zaporowe ustawione było na większość, wszystkie, lub prawie wszystkie częstotliwości składników zmiennych, zaś pasmo przepustowe bez modyfikacji pozostawiało kilka współczynników. W efekcie w obrazie pojawiły się bardzo duże zakłócenia.

5.2.3 Kodowanie bezstratne

Po przeprowadzeniu kwantyzacji współczynniki z bloku 8x8 ustawiane są w wektor długości 64, zgodnie z porządkiem zygzakowatym - patrz Tabela 4. Dzięki temu współczynniki są ustawione zgodnie z rosnącą częstotliwością przestrzenną. Tak przygotowane dane można poddać ostatniemu etapowi kompresji - kodowaniu entropijnemu.

Kodowanie bezstratne powyższego wektora polega na przekształceniu w sekwencję symboli (postać pośrednią), a następnie na zakodowaniu kodami Huffmana lub kodami arytmetycznymi. Pośrednia postać polega na tym, że każdy niezerowy składnik zmienny reprezentuje się parą symboli: (R, L), gdzie R to liczba poprzedzających składników zmiennych równych 0, natomiast L to jego wartość.

Jeśli chodzi o składniki stałe, to koduje się je w inny sposób. Jak już wspomnieliśmy w podrozdziale 4.4.3 są one między sobą silnie skorelowane, dlatego stosuje się predykcję DPCM, zaś jako predyktora używa się poprzedniego składnika stałego.[17] Stąd w postaci pośredniej zapisuje się błąd predykcji tego składnika. Następnie postać pośrednia kodowana jest kodami zmiennej długości i sygnał przekazywany jest na wyjście.

[17]Lub też kilku współczynników - np. tego po lewej, tego sąsiadującego bezpośrednio powyżej oraz dwóch powyżej sąsiadujących po przekątnej.

5.3 Tryb progresywny

5.3.1 Wyznaczanie widma

Ten etap jest taki sam jak w przypadku trybu sekwencyjnego, patrz podrozdział 5.2.1.

5.3.2 Kwantyzacja próbek widma

Ten etap jest taki sam jak w przypadku trybu sekwencyjnego, patrz podrozdział 5.2.2.

5.3.3 Kodowanie bezstratne

Zasadnicza różnica między trybem sekwencyjnym, a trybem progresywnym polega na wielokrotnym kodowaniu entropijnym danych, podczas gdy w trybie sekwencyjnym dane były kompresowane entropijnie w jednym przebiegu.

Dostępne są trzy tryby progresywne:

1. **Przez selekcję spektralną** - współczynniki dzieli się na kilka pasm, a następnie koduje się po kolei każde pasmo oddzielnie. Na przykład pierwsze pasmo to pierwszy współczynnik (w porządku zygzakowatym), drugie pasmo to współczynniki 2 i 3, trzecie to współczynniki 4, 5 i 6; czwarte to współczynniki 7 do 10 itd.

2. **Przez aproksymację sukcesywną** - najpierw koduje się najbardziej znaczące bity współczynników DCT, a w kolejnych przebiegach te coraz mniej znaczące. Na przykład [4, s. 11] najpierw koduje się wszystkie współczynniki DCT podzielone przez 4, w kolejnym przebiegu wszystkie współczynniki DCT podzielone przez 2, a w ostatnim wszystkie współczynniki DCT w pełnej dokładności.

3. **Mieszany progresywny** - połączenie dwóch powyższych metod (możliwe przeplatanie etapów np. podział na pasma i aproksymacja każdego z nich).

5.4 Tryb bezstratny

Składa się z dwóch etapów: predykcji oraz kodera entropijnego. Najpierw przeprowadza się predykcję wartości piksela na podstawie jego poprzedników, wylicza się różnicę między wartością przewidywaną piksela, a jego rzeczywistą wartością, a następnie ową różnicę poddaje się kodowaniu entropijnemu.

Tryb bezstratny redukuje ilość pamięci potrzebnej do zapisania obrazu średnio o połowę [4, s. 15].

5.5 Tryb hierarchiczny

Jak już wcześniej wspomnieliśmy - umożliwia kompresję w wielu rozdzielczościach. Ogólny algorytm wygląda następująco:

1. Zmniejszenie rozdzielczości obrazu do ustalonego najniższego poziomu
2. Kompresja zmniejszonego obrazu jednym z powyższych trybów, a następnie jego dekompresja - w ten sposób uzyskujemy predyktor
3. Podwajamy rozdzielczość predyktora odpowiednim filtrem
4. Obniżamy rozdzielczość oryginału do rozdzielczości predyktora, obliczamy różnicę, którą kodujemy w jednym z powyższych trybów i zapisujemy otrzymany sygnał na wyjście
5. kroki 3 i 4 powtarzamy tak długo, aż zakodujemy obraz w pełnej rozdzielczości

W ten sposób otrzymamy wspomnianą sekwencję ramek, dzięki której duży obraz przesyłany przez wolne łącze będzie widoczny od razu, ale jego jakość będzie się stopniowo poprawiać (stosownie do ilości otrzymanych ramek).

5.6 Struktura skompresowanych danych

Syntaktyka standardu JPEG to dwie grupy danych: dane właściwe zakodowane entropijnie oraz informacje nagłówkowe niezbędne do rozkodowania

sygnału, takie jak tabele kodów, tabele kwantyzacji itp. Stosuje się trzy formaty danych [10, s. 21]:

1. Format wymiany (interchange format) - służy do zapisu danych kodowanych entropijnie wraz z wszystkimi danymi niezbędnymi do zdekodowania sygnału. Format ten stosuje się do wymiany danych między różnymi programami.

2. Skrócony format skompresowanych danych (abbreviated format) - jak wyżej, z tą różnicą, że nie zawiera wszystkich tablic wymaganych do dekodowania.

3. Skrócony format specyfikacji tablic danych (abbreviated format for table-specification data) - zawiera wyłącznie tablice, które po umieszczeniu w dekoderze mogą posłużyć do zdekodowania kilku obrazów.

Definiuje się podstawową jednostkę danych dla danych bez przeplotu - jest to minimalna jednostka poddawana kodowaniu (zwykle blok 8x8).

Standard nie specyfikuje rozszerzenia pliku ani formatów. Dwa najczęściej stosowane formaty to JFIF wykorzystujący format wymiany oraz EXIF - stosowany w fotografii, umożliwia zapis różnych dodatkowych informacji o zdjęciu.

Istnieją rozszerzenia standardu JPEG (ujęte w normie ISO 10918-3) obejmujące [1, s. 374]:

1. wprowadzenie skalowania tablic kwantyzacji dla każdego bloku

2. możliwość kodowania skalowalnego na fragmencie sygnału

3. kodowanie dużych obrazów mniejszymi blokami

4. wprowadzenie formatu SPIFF (Still Picture Interchange File Format)

Są one jednak rzadko używane.

6 Inne metody kodowania

Alternatywą dla algorytmu JPEG bazującego na dyskretnej transformacie kosinusowej, jest JPEG2000 - bazujący na dyskretnej transformacie falkowej. Standard przyjęto w roku 2000, norma ISO 15444-1. Szczegółowe informacje na ten temat można znaleźć w: [1, s. 374], [8, s. 77]. Do dziś standard ten nie przyjął się tak szeroko jak JPEG.

Kolejnym standardem wartym wspomnienia jest JPEG XR, specyfikowany normami ISO 29199-1, ISO 29199-2, ISO 29199-3. Szczegóły można znaleźć w [1, s. 385].

Warto też wspomnieć o innych standardach: DjVu [2, s. 248], JBIG [2, s. 239], JBIG2 [2, s. 247], czy wreszcie o PNG [2, s. 191], TIFF czy GIF.

Literatura

[1] Domański, M., „Obraz cyfrowy. Reprezentacja, kompresja, podstawy przetwarzania. Standardy JPEG i MPEG.", Wydawnictwo Komunikacji i Łączności, 2010

[2] Przelaskowski, A., „Kompresja danych. Podstawy. Metody bezstratne. Kodery obrazów.", Wydawnictwo BTC, 2005

[3] Owczarz-Dadan, A., „ABC fotografii cyfrowej i obróbki zdjęć", Wydawnictwo Helion, 2006

[4] Krupiczka, A., „Standardy kompresji obrazów - studium porównawcze (JPEG, H.261, T.120, MPEG-1, MPEG-2)", Instytut Podstaw Informatyki Polskiej Akademii Nauk, 1996

[5] Dobosz, M. i Derk, T., Wykład specjalizacyjny - Multimedia 2, http://edu.pjwstk.edu.pl/wyklady/wspmu2/scb/main65.html

[6] Adamczak, M., „Optymalizacja metod redukcji obserwacji fotometrycznych CCD." Praca magisterska., Obserwatorium Astronomiczne, Uniwersytet im. Adama Mickiewicza w Poznaniu, 1999, http://vesta.astro.amu.edu.pl/Education/ccd2/node8.html

[7] Tadeusiewicz, R., Korohoda, P. „Komputerowa analiza i przetwarzanie obrazów", Wydawnictwo Fundacji Postępu Telekomunikacji, 1997, http://winntbg.bg.agh.edu.pl/skrypty2/0098/komputerowa_analiza.pdf

[8] Paluszyński, M., „Wstęp do analizy falkowej z zastosowaniami w matematyce finansowej", Skrypt to wykładu Wstęp do zastosowań analizy falkowej, 2011, http://www.math.uni.wroc.pl/~mpal/academic/2012/falki.pdf

[9] Olshausen, B., „Aliasing", 2000, http://redwood.berkeley.edu/bruno/npb261/aliasing.pdf

[10] International Telecomunication Union, „Information technology - digital compression and coding of continuous-tone still images - requirements and guidelines.”, 1993,
http://www.w3.org/Graphics/JPEG/itu-t81.pdf

Spis rysunków

Spis tabel