**Kodek Obrazu Wykorzystujący Asymetryczny System Numeryczny**

Artur Tkaczyk1

1 Politechnika Warszawska, plac Politechniki 1, 00-661 Warszawa

**Streszczenie.** Niniejszy artykuł omawia temat modyfikacji kodera oraz dekodera entropijnego w implementacji kodeka obrazu opartego na standardzie JPEG2000. Moim zadaniem jest podmiana kodowania arytmetycznego wykorzystywanego w oryginalnej implementacji na asymetryczny system numeryczny (ANS). Na początku omówię podstawy teoretyczne na temat zarówno kompresji, a także kodowania entropijnego. Omawiając kodowanie entropijne, przedstawię podstawowe algorytmy kodowania entropijnego. Na koniec części teoretycznej przedstawię pokrótce standard JPEG2000. Następnie omówię modyfikacje dokonane przeze mnie na oryginalnej implementacji. N a końcu przetestuję poprawność nowego kodera i dekodera oraz porównam ich działanie z oryginalną wersją.

**Słowa kluczowe:** kodowanie entropijne, ANS, kodowanie arytmetyczne, JPEG2000, kompresja obrazu.

# **Wstęp**

ANS jest nowym sposobem kodowania łączącym ze sobą wysoki stopień kompresji jak przy kodowaniu arytmetycznym oraz szybkość działania jak przy kodowaniu Huffmana. W ramach artykułu dokonam modyfikacji sposobu kodowania i dekodowania entropijnego w implementacji kodeka obrazu zgodnego ze standardem JPEG2000. Po skończeniu implementacji, przeprowadzę serię testów w celu określenia, czy otrzymano oczekiwany rezultat. Testy będą składały się z trzech części sprawdzających kolejno: poprawność, stopień kompresji oraz szybkość działania. Otrzymane rezultaty ocenię w porównaniu z oryginalną implementacją opartą na kodowaniu arytmetycznym.

Artykuł podzielony jest na sekcje:

* Podstawy teoretyczne kompresji danych – w tej sekcji omówię, czym jest kompresja i jakie są jej typy, czym jest entropia i jak się ją oblicza oraz czym jest kodowanie entropijne.
* Metody kodowania entropijnego – w tej sekcji przedstawię metody kodowania entropijnego: kodowanie Huffmana, kodowanie arytmetyczne, ANS, CABAC.
* Standard kompresji obrazów JPEG2000 – w tej sekcji ogólnikowo omówię standard JPEG2000.
* Implementacja – w tej sekcji omówię moją implementację kodera i dekodera ANS
* Testowanie – w tej sekcji omówię wszystkie przeprowadzone przeze mnie testy, mające na celu ocenienie poprawności, stopnia kompresji i szybkości nowej implementacji oraz przedstawię wyniki wraz z ostatecznymi wnioskami
* Podsumowanie - krótkie podsumowanie artykułu

# Podstawy teoretyczne kompresji danych

Szybki rozwój technologiczny pociąga za sobą ciągły wzrost ilości informacji przechowywanych na świecie i szacuje się, że proces ten będzie tylko przyśpieszał. Ponadto wzrosło zapotrzebowanie na szybkie przesyłanie sporych rozmiarów danych na przykład w celu przeprowadzenia transmisji wideo na żywo. Tak więc aby sprostać wymaganiom konieczne było ulepszenie metod przechowywania, przesyłania oraz manipulacji ogromną ilością danych. Aby usprawnić te procesy, tworzone są pamięci masowe o większej pojemności oraz łącza o większej przepustowości. Jednak oprócz tych znaczących usprawnień, wymagany jest również rozwój metod kompresji danych.



**Rys. 1.** Ilość danych na świecie [5]

Bardzo ważnym typem danych we współczesnych systemach informatycznych są obrazy. W samym tylko 2015 roku wykonano szacunkowo bilion obrazów cyfrowych [6]. W obrazach występuje dwukierunkowa korelacja między wartościami próbek, ponadto ze względu na typ obrazu stosuje się inne metody kompresji (na przykład obrazy naturalne kompresuje się w inny sposób niż obrazy medyczne, czy grafiki komputerowe).

Kompresja danych to proces przekształcania pierwotnej reprezentacji danych w reprezentację o mniejszej ilości bitów, odwrotny proces nazywany jest dekompresją. Wyróżnia się kompresje: stratne i bezstratne. W przypadku kompresji bezstratnej, w trakcie procesu kompresji danych nie tracimy żadnych informacji, co niestety pozwala na kompresje danych tylko do pewnego stopnia. W przypadku kompresji stratnej tracimy pewną część informacji, ale za to jesteśmy w stanie skompresować dane w większym stopniu. Celem kompresji stratnej jest uzyskanie danych, których reprezentacja (w postaci obrazu lub pliku audio, wideo) jest wystarczającej jakości, a więc opuszcza się dane mało istotne dla ludzkiej percepcji [1].

Kodowanie entropijne ma na celu usunięcie nadmiarowości z ciągu danych wejściowych, co jest osiągane przez przypisanie znakowi lub ciągowi znaków ciąg bitów, w taki sposób, by wielkość danych była jak najbardziej zbliżona do entropii. Nie można jednak osiągnąć wyniku mniejszego od wyliczonej entropii. Kodowanie entropijne wykorzystywane jest zwykle jako ostatni etap kompresji stratnej. Do wyliczenia entropii danych stosuje się wzór:

*H =* log2 *N =* log*2*(1 / *pN*)(1)

# Metody kodowania entropijnego

## **Kodowanie Huffmana**

Kodowanie Huffmana jest metodą kodowania entropijnego stworzoną w 1952 roku przez Davida Huffmana i opublikowaną w pracy pod tytułem „A Method for the Construction of Minimum-Redundancy Codes”. Polega ona na znalezieniu optymalnych kodów przedrostkowych poprzez zbudowanie na podstawie alfabetu znaków i jego rozkładu prawdopodobieństw, drzewa Huffmana [2]. Kodowanie Huffmana cechuje przede wszystkim szybkość działania oraz prostota implementacji. Niestety stopień kompresji dla tej metody jest względnie niski, a wielkość danych wyjściowych zbliża się do wartości entropii, tylko gdy prawdopodobieństwa występowania znaków zbliżone są do potęg dwójki.

## Kodowanie arytmetyczne

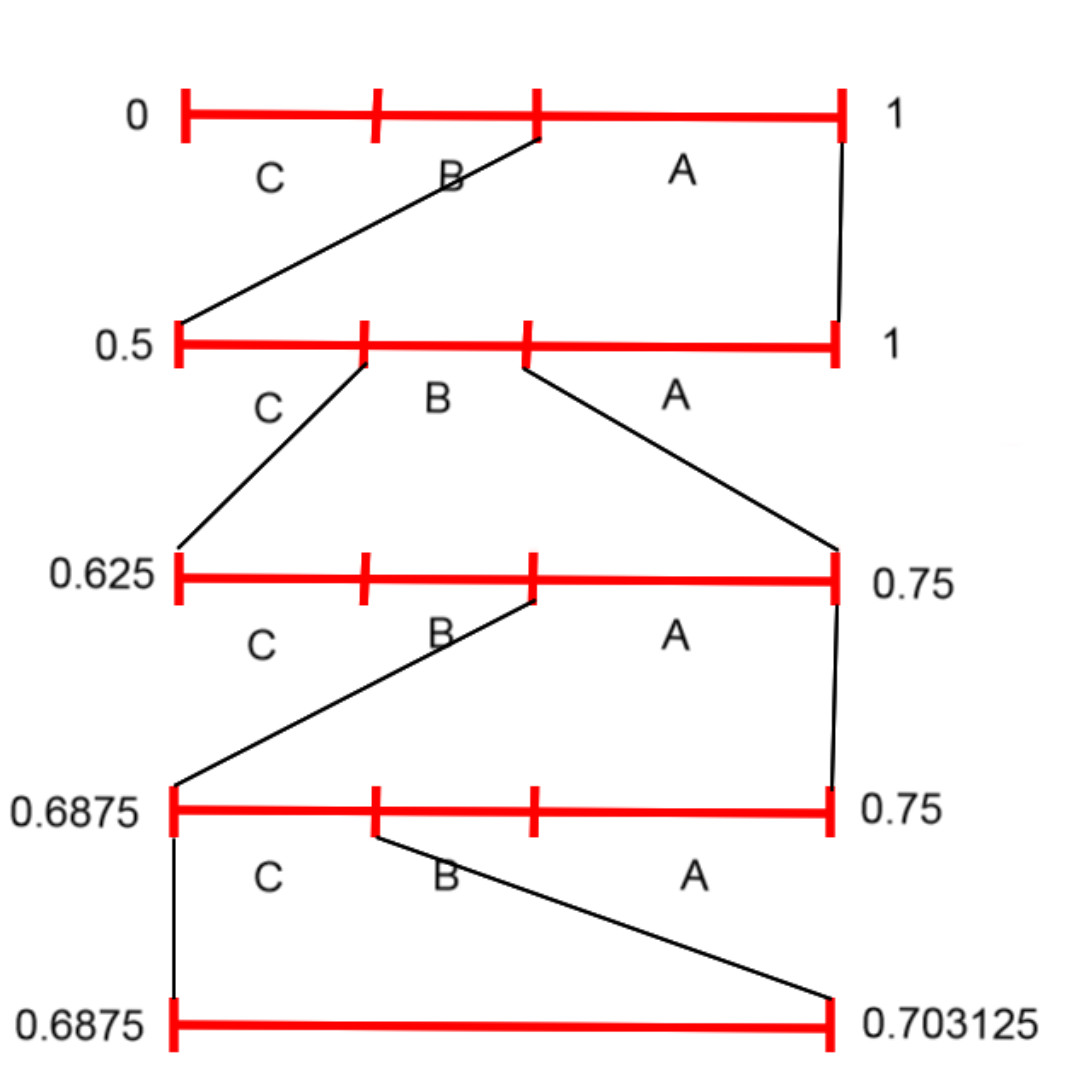
Wadą kodowania Huffmana jest to, że znaki alfabetu kodowane są za pomocą pełnej liczby znaków, a więc aby ulepszyć proces kodowania entropijnego, trzeba było znaleźć sposób na zakodowanie pojedynczego znaku za pomocą ułamków bitów. Kodowanie arytmetyczne jest metodą kompresji bezstratnej opracowanej około 1960 roku przez Petera Eliasa. Zapewnia on, że wielkość zakodowanych danych zbliżona jest do entropii. Poprawa ta wiąże się jednak ze spadkiem szybkości działania algorytmu.

Koder arytmetyczny przechowuje aktualny stan za pomocą przedziału, a więc w postaci dwóch wartości oznaczających dolną granicę przedziału oraz jego długość. Stan ten stale zmienia się wraz z procesem kodowania znaków przeprowadzanym przez koder, ale przedział ten zawsze zawiera się wewnątrz przedziału [0, 1). Wartość dowolnej liczby zawierającej się wewnątrz tego stanu, wraz z informacją o rozkładzie prawdopodobieństwa występowania poszczególnych znaków alfabetu w pełni wystarczy do przeprowadzenia procesu odwrotnego, a więc do poprawnego zdekodowania ciągu symboli. Dekoder podobnie jak koder, również przechowuje stan przedziału, który odpowiednio modyfikuje w trakcie działania całego procesu [7].

Na początku procesu kodowania inicjuje się stan kodera na przedział [0, 1). Na początku każdego kroku kodowania dzieli się aktualny przedział na podprzedziały o długościach wprost proporcjonalnych do rozkładu prawdopodobieństw znaków alfabetu wejściowego. Następnie pobiera się z wejścia następny znak do zakodowania i jako następny stan kodera ustawia się podprzedział odpowiadający kodowanemu znakowi. Po zakodowaniu ostatniego znaku zwracamy na wyjście kodera liczbę zmiennoprzecinkową zawierającą się wewnątrz ostatecznego przedziału.

Zaletą kodowania dla tej metody jest to, że symbole są zwracane z dekodera dokładnie w tej samej kolejności co były one kodowane, co sprawia między innymi to, że nie ma potrzeby buforowania danych na wejściu, a następnie odwracania bufora.

Na początku procesu dekodowania inicjuje się stan kodera na przedział [0, 1). Na początku każdego kroku dekodowania dzieli się aktualny przedział na podprzedziały o długościach wprost proporcjonalnych do rozkładu prawdopodobieństw znaków. Następnie sprawdza się do którego z tych podprzedziałów należy liczba zmiennoprzecinkowa pobrana z wejścia dekodera. Wybrany przedział ustawia się na aktualny stan dekodera, a znak z nim związany zwracany jest na wyjście.



**Rys. 2.** Przykład działania kodowania arytmetycznego. Kodowany jest tu ciąg ABAC

Szczególną odmianą kodera arytmetycznego jest kontekstowo-adaptacyjny binarny koder arytmetyczny, w skrócie CABAC. Koder tego typu otrzymuje na wejście stowarzyszone ze sobą wartości: bit i kontekst. Dla każdego kontekstu wyliczany jest osobny model prawdopodobieństwa. Oznacza to, że dla każdego poszczególnego kontekstu rozkład prawdopodobieństwa między oba symbolami może być diametralnie inny. Adaptacyjność kodera oznacza natomiast, że rozpoczyna on swoje działanie z domyślnymi, początkowymi wartościami rozkładu prawdopodobieństw dla każdego z poszczególnych kontekstów. Wartości te jednak zmieniają się w trakcie procesu kodowania [9].

## Asymetryczny System Numeryczny (ANS)

Asymetryczne kodowanie numeryczne (z angielskiego Assymetrical Numeral Systems, w skrócie ANS) jest nową metodą kodowania entropijnego opracowaną przez dr Jarosława Dudę. Łączy ona w sobie stopień kompresji porównywalny to tego, otrzymanego w wyniku kodowana arytmetycznego oraz szybkość działania kodowania Huffmana [4].

Asymetryczne kodowanie numeryczne przechowuje informację w pojedynczej wartości. Nowa wartość stanu kodera po zakodowaniu symbolu o prawdopodobieństwie *p* wynosi:

*x’* ≈ *x* / *p* (2)

Po dekodowaniu nowa wartość stanu dekodera wynosi:

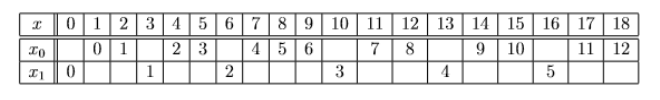
*x’* ≈ *x* \* *p* (3)

Ze wzoru na entropię widać, że w wyniku podzielenia stanu kodera przez prawdopodobieństwo znaku otrzymujemy nową wartość posiadającą entropię będącą sumą entropii poprzedniego stanu i entropii kodowanego znaku:

log*2*(*x* / *p*) = log*2*(*x*) + log*2*(1 / *p*)(4)

Jedną z dodatkowych cech różniących go od kodowania arytmetycznego jest to, że pierwszy zakodowany znak będzie ostatnim zdekodowanym. W wyniku tego pojawia się konieczność buforowania danych wejściowych oraz utrudnia zastosowanie ANS w wersji adaptacyjnej. Biorąc pod uwagę to, że wartość stanu kodera, będzie z czasem rosła, należy zastosować renormalizację wartości, aby uniemożliwić wzrost wartości do nieskończoności. Podczas renormalizacji wyrzuca się mniej znaczące bity stany kodera, tak aby jego wartość znów mieściła się w dopuszczalnych granicach. Wysunięte bity wstawia się na wyjście kodera. Po zakodowaniu ciągu znaków, należy jeszcze odwrócić ciąg bitów.

W przypadku wariantu uABS (uniform Asymmetric Binary System) stosowany jest dwuelementowy alfabet wejściowy. Dzielimy zbiór liczb naturalnych na dwa podzbiory, których gęstość jest zależna od rozkładu prawdopodobieństw znaków wejściowych.



**Rys. 3.** Przykładowy podział zbioru liczb naturalnych dla uABS

Dla kodowania symbolu 0:

*x’* = ⌈(*x* + 1) / (1 - *q*)⌉ - 1(5)

Dla kodowania symbolu 1:

*x’* = ⌊*x* / *q*⌋ (6)

Wzory wykorzystywane do procesu dekodowania wyglądają następująco:

Dekodowanie symbolu:

*s* = ⌈(*x* + 1)*q*⌉ - ⌈*xq*⌉(7)

W zależności od zwróconej wartości s stosuje się jedno z dwóch równań.

Dla s = 0:

*x*0 *= x* -⌈*xq*⌉(8)

Dla s = 1:

*x*1 *=* ⌈*xq*⌉(9)

Wariant tablicowy asymetrycznego kodowania numerycznego, czyli tANS pozwala na umieszczenie wyliczonych przy użyciu powyższych wzorów wartości w tablicy. Pozwala to na otrzymanie znacznie szybszego działania, biorąc pod uwagę to, że jedyne operacje wykonywane w trakcie samego kodowania będą polegać na pobraniu określonych wartości z tablicy.

W przypadku kodowania wyróżniamy trzy wartości, które są umieszczane w tablicy:

* następny stan kodera
* ilość wysuwanych na wyjście bitów
* wartości wysuwanych bitów

Wartości te zależne są od dwóch wartości: aktualnego stanu kodera i kodowanego znaku.

Natomiast dla procesu dekodowania mamy wartości:

* następny stan dekodera przed renormalizacją
* ilość pobieranych bitów z wejścia
* zdekodowany znak

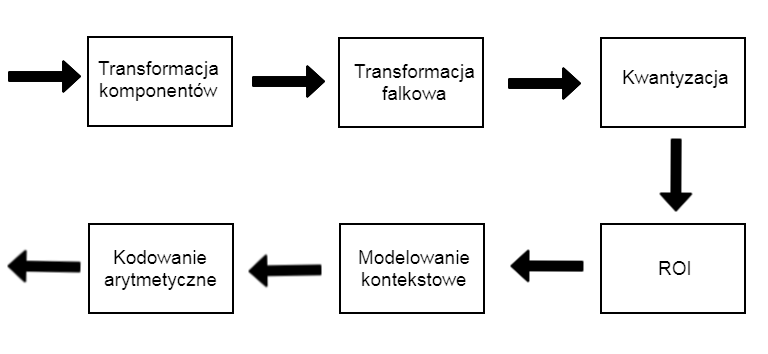
Tutaj w celu pobrania odpowiednich wartości z tablicy potrzebujemy jedynie stan dekodera.

# Standard kompresji obrazów JPEG2000

Standard kompresji obrazów JPEG2000 został opracowany w 2000 roku poprzez uzupełnienie istniejącego standardu JPEG (*Joint Photographic* *Experts* *Group*) o nowe techniki kompresji [3].

Cechy charakterystyczne:

* Wyższa jakość obrazu dla wysoce skompresowanych danych. JPEG2000 oferuje wyższą jakość obrazów, jednak różnica ta w stosunku do standardu JPEG, jest najlepiej widoczna dla obrazów wysoce skompresowanych.
* Różne ilości bitów na próbkę obrazu. Można stosować JPEG2000 zarówno dla obrazów z jednym bitem na próbkę, jak i obrazów wielokolorowych
* Progresywna kompresja obrazu. Jest to cecha istotna w szczególności dla transmisji obrazu przez wolne łącze internetowe. Na początku wysyłany jest obraz o niskiej jakości, a następnie wysyłane są kolejne warstwy poprawiające jakość wcześniej wysłanego obrazu.
* Możliwość kompresji bezstratnej. Pomimo, że JPEG2000 przeznaczony jest głównie do kompresji stratnej, to jest również możliwość stosowania kompresji bezstratnej
* Prosty dostęp do dowolnego regionu obrazu.
* Dekompresja na wiele różnych sposobów z jednego skompresowanego obrazu. Standard JPEG2000 pozwala na dekompresowanie jednego obrazu w wielu różnych trybach w zależności od aktualnych potrzeb.



**Rys. 4.** Schemat kodera JPEG2000

# Implementacja

Zmiany dokonywane będą na implementacji referencyjnej kodera i dekodera JJ2000 [10]. Procesem kontekstowego kodowania entropijnego zajmuje się klasa MQCoder w pakiecie ucar.jpeg.jj2000.j2k.entropy.encoder. Dekodowanie natomiast obsługiwane jest przez klasę MQDecoder w pakiecie ucar.jpeg.jj2000.j2k.entropy.decoder. Udało mi się przeprowadzić odpowiednie modyfikacje, dokonując zmian jedynie w tych dwóch klasach, nie zmieniając ich publicznych metod ani żadnych pozostałych klas w oryginalnej implementacji.

Zastosowałem ANS w wariancie tANS, operującym na alfabecie dwuelementowym. Tak więc w obydwu klasach pierwszym etapem było utworzenie tablic z wynikami, do których koder oraz dekoder mogły sięgać.

Ponieważ wynik kodowania znaku, a więc także wartości przetrzymywane w wyliczonych tablicach są zależne od wartości prawdopodobieństw dlatego pojawia się problem, ponieważ rozkładów prawdopodobieństw między bitami może być nieskończenie wiele, a więc zamiast korzystając bezpośrednio z prawdopodobieństw, korzystam ze skończonej ilości modeli prawdopodobieństw, które przypisane są do określonych wartości prawdopodobieństwa. Spadek stopnia kompresji spowodowany tym przybliżeniem jest stosunkowo mały. W referencyjnej implementacji również stosowane były modele prawdopodobieństw, a więc na daną chwilę postanowiłem skorzystać z nich, a więc ilość modeli oraz przybliżone wartości im przypisane pozostały na razie bez zmian. Na tą chwilę ilość modeli wynosi 47, ale w następnym etapie projektu sprawdzę jaka wartość będzie najbardziej optymalna.

Ponieważ korzystam z wersji ANS w postaci stablicowanej, a więc dopuszczalne stany kodera muszą się zawierać w granicach [2n, 2n+1 - 1]. Górna granica przedziału ma na celu zapewnienie, że wartość stanu nie będzie rosła w nieskończoność, co spowodowałoby problemy z przechowywaniem stanu kodera, a także drastycznie zwiększyłoby wielkość tablic kodowania. Żeby przeciwdziałać przekroczeniu przez wartość stanu tej granicy stosowana jest renormalizacja stanu przed kodowaniem, które doprowadzi do przekroczenia górnej granicy. Dolna granica natomiast konsekwencją zastosowania renormalizacji. Gdyby dolna granica znajdowała się niżej lub w ogóle by nie istniała pojawiłaby się niejednoznaczność dekodowania. Na daną chwilę ustaliłem przedział dopuszczalnych stanów na [1024, 2047], co daje łącznie 1024 dopuszczalne stany. Zwiększenie tej wartości nie zmniejszy ilości stosowanych renormalizacji, a więc nie przyśpieszy kodowania, za to znacznie zwiększy znacznie wielkość tablic kodowania, a więc zapotrzebowanie na pamięć. Ilość stanów nie może jednak również zbyt mała, ponieważ nie pozwoliłoby to stosowanie modeli prawdopodobieństw, w których jest duża dysproporcja między częstością występowania poszczególnych bitów, co mogłoby obniżyć stopień kompresji. Z moich obserwacji obecna ilość stanów jest w pełni wystarczająca bez większego wpływu na kompresję danych.

Do utworzenia tablicy przechowywanej w koderze stosuję standardowe wzory dla ANS, natomiast podczas tworzenia tablic dla dekodera wykorzystuję wytworzoną wcześniej tablicę do kodowania.

Tablica wykorzystywana do kodowania jest czterowymiarową tablicą przechowującą wartości: następny stan kodera, ilość bitów zwracanych na wyjście oraz wartość zwróconych bitów. Wartości te zależą od stosowanego modelu prawdopodobieństwa, aktualnego stanu kodera oraz kodowanego bitu. Proces kodowania ogranicza się więc do pobrania odpowiednich wartości z tablicy, a następnie na ich podstawie zaktualizować stan kodera i wypisać na wyjście bity po renormalizacji.

ANS w przeciwieństwie do kodowania arytmetycznego dekoduje znaki od końca, a więc ostatni zakodowany znak będzie pierwszym zdekodowanym. Stwarza to spory problem, ponieważ cecha ta utrudnia zachowanie adaptacyjności kodera. Ponadto, aby bity zwrócone przez dekoder były w odpowiedniej kolejności, trzeba dane wejściowe kodera zbuforować na następnie bufor ten przed kodowaniem odwrócić. Postanowiłem więc podczas buforowania danych na wejściu kodera wyliczać prawdopodobieństwa występowania poszczególnych bitów dla każdego kontekstu. Ponieważ stosuję modele prawdopodobieństwa zamiast zwykłych prawdopodobieństw, dlatego następnie muszę wybrać taki model, który jest najbliższy wyliczonej wartości. Dla całego kodowanego bloku danych stosowane są wyliczone modele prawdopodobieństw, które następnie dołączane są na początku zakodowanego bloku danych, tak aby dekoder wiedział z jakimi wartościami kodowane były dane.

Zgodnie ze standardem JPEG2000, w wyjściowym strumieniu bitowym stosowane są znaczniki oznaczające specjalne części strumienia, takie jak: początek pakietu, koniec nagłówka pakietu. Wszystkie znaczniki mieszczą się w granicach od 0xFF90 do 0xFFFF. W związku z tą cechą standardu zakodowane bloki danych nie mogą zawierać tych wartości. Rozwiązuję ten problem poprzez dodanie bitu 0 po każdym wykrytym bajcie 0xFF.

Oprócz odwrócenia bitów na wejściu kodera, potrzebne jest również odwrócenie wynikowego ciągu, tak aby ostatnie bity zwrócone z kodera, były pierwszymi wstawionymi na wejście dekodera.

Tablica wykorzystywana do dekodowania jest trzywymiarową tablicą przechowującą wartości: następny stan dekodera przed renormalizacją, ilość bitów pobieranych z wejścia oraz wartość zdekodowanego bitu. Wartości te zależą od stosowanego modelu prawdopodobieństwa oraz aktualnego stanu dekodera. Proces dekodowania ogranicza się więc do pobrania odpowiednich wartości z tablicy, a następnie pobraniu bitów wejścia i wstawieniu ich na najmniej znaczących pozycjach pobranego stanu. Po wszystkim aktualizujemy wyliczony stan i zwracamy pobrany z tablicy bit.

Konteksty są obiektami, w ramach których przetwarzane są bity, Zawierają informacje dotyczące stosowanego modelu prawdopodobieństwa oraz który z bitów jest mniej prawdopodobny do wystąpienia w ramach danego kontekstu.

# Testowanie

Na dany moment ukończona jest wstępna wersja kodeka, implementacja będzie jeszcze udoskonalana. Kodek dopuszcza trzy formaty obrazów do kompresji: PPM, PGM i PGX. Dla każdego z tych formatów przeprowadziłem po kilka testów poprawności, które dały pozytywny wynik. Porównanie wielkości danych wyjściowych dla zmodyfikowanego i oryginalnego kodeka nie pokazuje dużej różnicy między nimi.

**Tabela 1.** Tabela z wynikami kompresji obrazów

|  |  |  |
| --- | --- | --- |
| Obraz | Oryginalna implementacja (w bajtach) | Zmodyfikowana implementacja (w bajtach) |
| jezioro.ppm | 31632 | 32081 |
| latarnia.ppm | 265059 | 264984 |
| tree.ppm | 593661 | 587352 |
| wildlife.ppm | 272805 | 278919 |
| zachod.ppm | 299414 | 301677 |
| panda.ppm | 77795 | 78746 |
| lew.ppm | 193373 | 198307 |
| balwan.ppm | 357944 | 362909 |
| tygrys.ppm | 375716 | 373602 |
| natura.ppm | 2381601 | 2363245 |

Porównanie szybkości działania obu kodeków wykazuje, że oryginalna implementacja jest na daną chwilę szybsza.

# Podsumowanie

Na obecną chwilę stosowane są głównie trzy metody kodowania entropijnego: kodowanie Huffmana (szybki w działaniu, prosta implementacja, niski stopień kompresji), kodowanie arytmetyczne (wolny w działaniu, wysoki stopień kompresji) oraz ANS (szybki w działaniu, wysoki stopień kompresji). Wstępna wersja kodeka jest gotowa, ale wymaga jeszcze poprawek. Testy wykazują poprawność działania kodeka, zbliżony stopień kompresji do oryginalnej implementacji. Porównanie szybkości działania pokazuje, że moja implementacja jest wolniejsza, co może być jednak spowodowane tym, że oprogramowanie jest zaimplementowane w języku Java.

# **Bibliografia**

1. Przelaskowski, A.: Kompresja danych: podstawy, metody bezstratne, kodery obrazów. BTC (2005).
2. Huffman, D.: A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the IRE, vol. 40 (1952)
3. Marcellin, M., Taubman, D.: JPEG2000 Image Compression Fundamentals, Standard and Practise. Springer Science+Business Media (2002)
4. Duda, J.: Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding. ArXiv:1311.2540 (2013).
5. Strona internetowa CM., <https://www.cm.com/blog/the-more-personalized-the-better-why-you-need-a-customer-data-platform/>, last accessed 2020/06/03.
6. Strona internetowa Business Insider., <https://businessinsider.com.pl/international/people-will-take-12-trillion-digital-photos-this-year-thanks-to-smartphones/bwd8pxp>, last accessed 2020/06/03.
7. Langdon, G.: An Introduction to Arithmetic Coding. [IBM Journal of Research and Development](https://dl.acm.org/journal/ibmj) (1984)
8. Strona internetowa ScienceDirect., <https://www.sciencedirect.com/topics/computer-science/arithmetic-coding>, last accessed 2020/06/03.
9. Strona internetowa ResearchGate., <https://www.researchgate.net/figure/CABAC-block-diagram-from-the-encoder-perspective-Binarization-context-modeling_fig1_290180658>, last accessed 2020/06/03.
10. Repozytorium Github., <https://github.com/Unidata/jj2000>, last accessed 2020/06/03.