| Instrukcja nr 4 | Temat: **Przetwarzanie sygnału w oparciu o transformatę falkową (wavelet)** | |
|---|---|---|
| Przedmiot: **Algorytmy i języki programowania** | | Wymiar: 2×LK 3×45 min. |
| Opracował: dr inż. Viktor Dashkiiev | Zespół Analizy Danych i Statystyki Stosowanej<br>Katedra Informatyki Stosowanej<br>Wydział Mechaniczny<br>Politechnika Krakowska | |

- **Cel zajęć**

badanie transformaty falkowej obrazu wizualnego z implementacją w języku C#

- **Wymagane oprogramowanie**

Microsoft Visual Studio, alternatywnie - każde środowisko programistyczne umożliwiające napisanie oraz
skompilowanie kodu w języku C#.

## *Część 1. Podstawy teoretyczne transformaty falkowej*

1.1. Podstawowe definicje

Ciągła transformata Fouriera (CFFT) jest niezbędnym narzędziem do analizy stacjonarnych sygnałów ciągłych. W tym przypadku sygnał jest rozkładany na bazę sinusów i cosinusów o różnych częstotliwościach. Liczba tych funkcji jest nieskończenie duża. Współczynniki konwersji oblicza się, obliczając iloczyn skalarny sygnału ze złożonymi wykładnikami:
Tak

$$F(\Omega) = \int_{-\infty}^{\infty} f(t) e^{-j\Omega t}\, dt\,,$$

gdzie f(t) to sygnał, a F(Ω) to jego transformata Fouriera.

### 1.2. Wprowadzenie do teorii transformaty falkowej

W ciągu ostatnich dwóch dekad na świecie wyłonił się i ukształtował nowy kierunek nauki związany z tzw. transformacją falkową. Słowo „falka” (wavelet – ang.), będące tłumaczeniem francuskiego „ondelette”, oznacza małe fale następujące po sobie. W wąskim sensie falki są rodziną funkcji uzyskanych przez skalowanie i przesuwanie jednej, matki, funkcji. W szerokim sensie falki są funkcjami, które mają dobrą lokalizację częstotliwości i których średnia wartość wynosi zero.

Transformata falkowa sygnału jednowymiarowego jest jego reprezentacją w postaci szeregu uogólnionego lub całki Fouriera po układzie funkcji bazowych (w tej pracy ortogonalnym)

$$\psi_{ab}(t) = |a|^{-1/2}\,\psi\left(\frac{t-b}{a}\right),$$

zbudowane z falki macierzystej (generującej) (t), która w wyniku operacji ma określone właściwości wskutek przesunięcia w czasie (b) i zmiany skali czasu (a).
Współczynnik $|a|^{-2}$ zapewnia niezależność normy tych funkcji od liczby skalującej ***a***.
Zatem dla danych wartości parametrów ***a*** i ***b*** funkcją $\psi_{ab}(t)$ jest falka, generowana przez falkę macierzystą $\psi(t)$.

### 1.3. Dyskretny transformat falkowy

W analizie numerycznej i funkcjonalnej dyskretne transformaty falkowe (DWT) odnoszą się do transformacji falkowych, w których falki są reprezentowane przez sygnały dyskretne (próbki).
Pierwszy DWT został wynaleziony przez węgierskiego matematyka Alfreda Haara. W przypadku sygnału wejściowego reprezentowanego przez tablicę $2^n$ liczb transformata falkowa Haara po prostu grupuje elementy po 2 i tworzy z nich sumy i różnice. Grupowanie sum odbywa się rekurencyjnie, tworząc kolejny poziom dekompozycji. Rezultatem jest różnica $2^n-1$ i $1$ suma całkowita.
Ten prosty DWT ilustruje ogólne użyteczne właściwości falek. Po pierwsze, transformację można przeprowadzić w operacjach $n \cdot \log 2$ (n). Po drugie, nie tylko rozkłada sygnał na pozory pasm częstotliwości (poprzez analizę go w różnych skalach), ale także reprezentuje dziedzinę czasu, czyli momenty wystąpienia określonych częstotliwości w sygnale. Razem te właściwości charakteryzują szybką transformatę falkową. Przyjmując warunek losowości sygnału ***X***, oblicza się gęstość widmową jego amplitud ***Y*** w oparciu o algorytm Yeatsa: macierz***Y***=macierz(±***X***), prawdziwe jest także odwrotne: macierz ***X***=macierz(±***Y***).
Najbardziej powszechny zestaw dyskretnych transformacji falkowych sformułowała belgijska matematyk Ingrid Dobieszy (Ingrid Daubechies) w 1988 roku. Opiera się na wykorzystaniu relacji rekurencji do obliczania coraz dokładnych próbek niejawnie określonej funkcji falki macierzystej, podwajającej rozdzielczość przy przejściu do następnego poziomu (skali). W swojej przełomowej pracy Daubechies wyprowadza rodzinę falek, z których pierwszą jest falka Haara. Od tego czasu zainteresowanie tą dziedziną szybko się rozwijało, co doprowadziło do powstania licznych potomków oryginalnej rodziny falek Daubechies.
Inne formy dyskretnej transformaty falkowej obejmują niezdziesiątkowaną transformację falkową (w której nie wykonuje się decymacji sygnału), transformatę Newlanda (w której ortonormalna baza falkowa jest wyprowadzana ze specjalnie

skonstruowanych filtrów kapeluszowych w dziedzinie częstotliwości). Wsadowe transformaty falkowe są również powiązane z DWT.
Inną formą DWT jest złożona transformata falkowa.
Dyskretna transformata falkowa ma wiele zastosowań w naukach przyrodniczych, inżynierii i (w tym stosowanej) matematyce. DWT jest najczęściej stosowana w kodowaniu sygnałów, gdzie właściwości transformacji są wykorzystywane w celu zmniejszenia redundancji w reprezentacji sygnałów dyskretnych, często jako pierwszy krok w kompresji danych. Dziś DWT zajmuje niezwykłe ważne miejsce we współczesnych technologiach informacyjnych, bowiem jest nieodzowną do kompresji, przesyłania, przechowywania i późniejszej dekompresji różnorakiej informacji, jak to: obrazów wizualnych, wideo- audio- itp.

### 1.4. Konwersja na jednym poziomie

DWT sygnału $\boldsymbol{x}$ uzyskuje się za pomocą zestawu filtrów. Najpierw sygnał przepuszcza się przez filtr dolnoprzepustowy o odpowiedzi impulsowej $\boldsymbol{g}$ i uzyskuje się splot:

$$y[n] = (x * g)[n] = \sum_{k=-\infty}^{\infty} x[k]g[n-k]$$

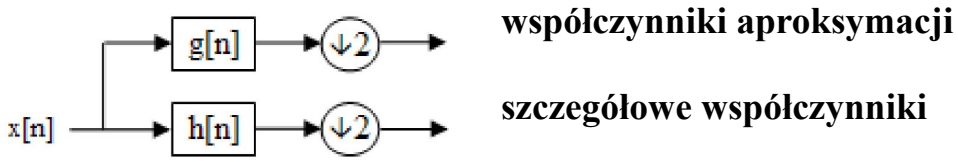
Jednocześnie sygnał jest rozkładany za pomocą filtra górnoprzepustowego $\boldsymbol{h}$. W wyniku są **szczegółowe współczynniki** (po filtrze górnoprzepustowym) i **współczynniki aproksymacji** (po filtrze dolnoprzepustowym). Te dwa filtry są ze sobą powiązane i nazywane są kwadraturowymi filtrami lustrzanymi (QMF).
Ponieważ połowa zakresu częstotliwości sygnału została przefiltrowana, zgodnie z twierdzeniem Kotelnikowa próbki sygnału można przezrzedzić dwukrotnie:

$$y_{\text{low}}[n] = \sum_{k=-\infty}^{\infty} x[k]g[2n-k]$$

$$y_{\text{high}}[n] = \sum_{k=-\infty}^{\infty} x[k]h[2n-k]$$

Rozkład ten zmniejszył o połowę rozdzielczość czasową z powodu przezrzedzenia sygnału. Jednakże każdy z powstałych sygnałów reprezentuje połowę pasma

częstotliwości sygnału pierwotnego, więc rozdzielczość częstotliwości została podwojona.




*Schemat rozkładu sygnału w DWT*

Korzystając się z operatora przerzedzania:

$$(y\downarrow k)[n] = y[kn]$$

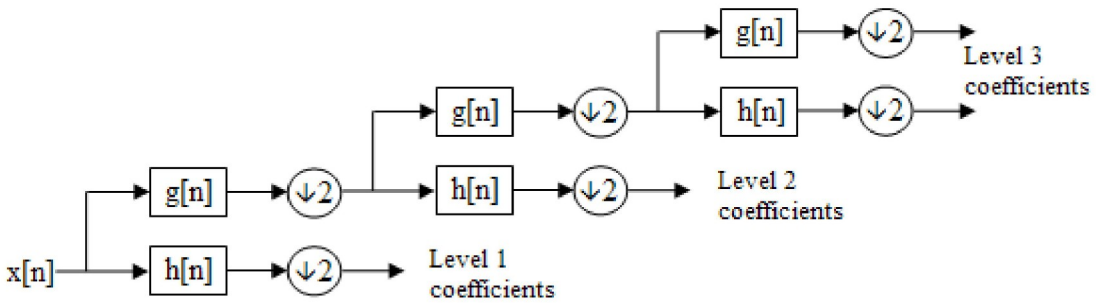
Powyższe sumy można w skrócie zapisać:

$$y_{low} = (x * g)\downarrow 2$$
$$y_{high} = (x * h)\downarrow 2$$

Obliczanie pełnego splotu ***x*g*** z późniejszym przerzedzeniem jest marnowaniem zasobów obliczeniowych.
**Schemat podnoszenia (lifting)** jest optymalizacją opartą na naprzemiennym wykonywaniu tych dwóch obliczeń.

### 1.5. Kaskadowanie i banki filtrów

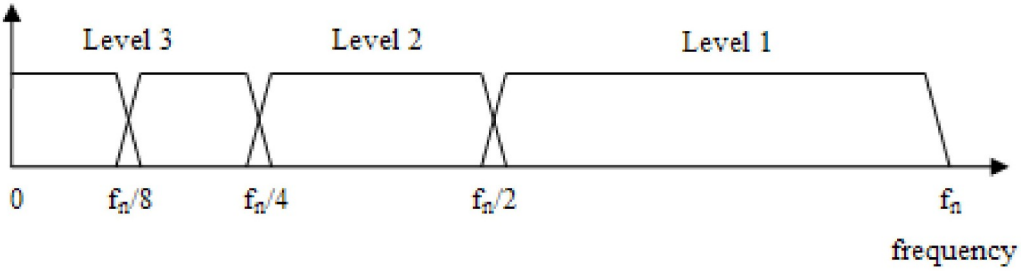
Rozkład ten można powtórzyć kilka razy, aby jeszcze bardziej zwiększyć rozdzielczość częstotliwości poprzez dalsze zmniejszenie współczynników po filtrowaniu dolnoprzepustowym i górnoprzepustowym. Można to przedstawić jako drzewo binarne, w którym liście i węzły odpowiadają przestrzeniom o różnych lokalizacjach czasowo-częstotliwościowych. To drzewo reprezentuje strukturę banku filtrów.




*Trzypoziomowy bank filtrów (grzebień)*

Na każdym poziomie powyższego diagramu sygnał jest rozkładany na niskie i wysokie częstotliwości. Ze względu na podwójne rozcieńczanie długość sygnału musi być wielokrotnością $2^n$, gdzie $n$ jest liczbą poziomów rozkładu.
Na przykład dla sygnału składającego się z *32* próbek o zakresie częstotliwości od *0* do $f_n$, trójpoziomowa dekompozycja da *4* sygnały wyjściowe w różnych skalach:

| Poziom | Częstotliwości | Długość sygnału |
|---|---|---|
| 3 | $0...f_n/8$ | 4 |
| | $f_n/8...f_n/4$ | 4 |
| 2 | $f_n/4...f_n/2$ | 8 |
| 1 | $f_n/2...f_n$ | 16 |




*Reprezentacja DWT w dziedzinie częstotliwości*

### 1.6. Algorytm Haara

Przykład szybkiej jednowymiarowej transformaty falkowej przy użyciu falki Haara dla tablicy danych początkowych o rozmiarze $2^N$ (odpowiednio liczba kaskad filtrów wynosi $N$) w języku C#:

```
public  static List<Double> DirectTransform(List<Double >  SourceList)
{
if
(SourceList.Count   == 1)
return
SourceList;
List<Double> RetVal  =  new List<Double>
();
List<Double> TmpArr   =  new List<Double>
();
for   (int  j   = 0;
j  <  SourceList.Count   - 1; j + = 2)
{
RetVal.Add((SourceList[j] -SourceList[j   +  1])   / 2.0);
TmpArr.Add((SourceList[j] + SourceList[j   +  1])   / 2.0);
}
RetVal.AddRange(DirectTrans form(TmpArr));
return RetVal;
}
```

Podobnie przykład odwrotnej transformaty falkowej:

```
public  static List<Double> InverseTransform(List<Doubl e>  SourceList)
{
if
(SourceList.Count   == 1)
return
SourceList;
List<Double> RetVal  =  new List<Double>
();
List<Double> TmpPart   =  new List<Double>
();
for   (int  i =
SourceList.Count  /   2;   i < SourceList.Count;
TmpPart.Add(SourceList[i]);
List<Double>
SecondPart =
InverseTransform(TmpPart);
for   (int  i  = 0;
i <  SourceList.Count  / 2; i++)
{
RetVal.Add(SecondPart[i] + SourceList[i]);
RetVal.Add(SecondPart[i] -SourceList[i]);
}
```

### 1.7. Dwuwymiarowa transformata falkowa

Przy opracowywaniu nowego standardu JPEG-2000 do kompresji obrazu było wybrano transformatę falkową. Sama transformacja falkowa nie kompresuje danych, ale pozwala na przekształcenie obrazu wejściowego w taki sposób, aby bez zauważalnego pogorszenia jakości obrazu, możesz zmniejszyć jego redundancję.

## 2. Wykorzystanie transformaty falkowej do kompresji i dekompresji obrazów wizualnych.

Niżej wyświetlony jest przykład kodu w języku C# dla kompresji i następnej dekompresji plików obrazów wizualnych.

**Uwaga!** Z oglądu na łatwość realizacji, program niżej pracuje tylko z plikami obrazów kwadratowych wielkość strony których równa się potęgi liczby 2

### 2.1. Przykład kodu

***Plik Program.cs***:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Drawing;
using System.Runtime.InteropServices;
using System.Drawing.Imaging;
namespace WaveleteCompression
{
   class Program
   {
      static void Main(string[] args)
```

```
        {

            // Plik z obrazem do kompresji (algorytm zezwala na kompresję
            // tylko obrazów kwadratowych wielkość strony których równa się potęgi liczby 2)
            string path = "C:\\Users\\Adresa pliku\\Nazwa pliku.png";

            // Kompresor
            // Uruchamiamy timer
            DateTime startTime = DateTime.Now;
            wvCompress c = new wvCompress();
            byte[] compressed = c.run(path);
            // Rozrachunok i wyświetlanie w konsoli czasu zapotrebowanego
            TimeSpan duration = DateTime.Now - startTime;
            Console.Write("Compressed: ");
            Console.WriteLine(duration.Seconds * 1000 + duration.Milliseconds + " ms");

            // Dekompresor
            // Uruchamiamy timer
            startTime = DateTime.Now;
            wvDecompress d = new wvDecompress();
            byte[] decompressed = d.run(compressed);
            duration = DateTime.Now - startTime;
            Console.Write("Decompressed: ");
            Console.WriteLine(duration.Seconds * 1000 + duration.Milliseconds + " ms");

            // Dekompresowany obraz
            Bitmap bitmap1 = BytesToBitmap(decompressed);
            // Zapis obrazu dekompresowanego
            bitmap1.Save(path + ".bmp", ImageFormat.Bmp);

            // Zapis w formacie RAW bez post-kompresji (żeby eksperymentować z post-
kompresją)
            FileStream f = new System.IO.FileStream(path + ".bmp.raw", FileMode.Create,
FileAccess.Write);
            f.Write(decompressed, 0, decompressed.Length);
            f.Close();

            string ret = Console.ReadLine();

        }

        public unsafe static Bitmap BytesToBitmap(byte[] data)
        {
            Size size = new System.Drawing.Size(512, 512);
            GCHandle handle = GCHandle.Alloc(data, GCHandleType.Pinned);
            Bitmap bmp = new Bitmap(size.Width, size.Height, size.Width * 3,
PixelFormat.Format24bppRgb, handle.AddrOfPinnedObject());
            handle.Free();
            return bmp;
        }

    }
```

```
}
```

***Plik wvCompress.cs***:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;
using System.IO.Compression;
using System.IO;

namespace WaveleteCompression
{

    class wvCompress
    {

        // Konstanty
        public const int WV_LEFT_TO_RIGHT = 0;
        public const int WV_TOP_TO_BOTTOM = 1;

        public byte[] run(string path)
        {

            // Ladujemy obraz z pliku
            Bitmap bmp = new Bitmap(path, true);

            // Przetwarzamy obraz załadowany do tablicy bajtów
            byte[, ,] b = this.BmpToBytes_Unsafe(bmp);

            // Zastosowanie wavelet'a
            byte[] o = this.Compress(b, bmp.Width, bmp.Height);

            // Zapis w formacie RAW bez post-kompresji
            FileStream f = new System.IO.FileStream(path + ".raw", FileMode.Create,
FileAccess.Write);
            f.Write(o, 0, o.Length);
            f.Close();

            // Kompresja otrzymanej tablicy procedurą Gzip i zapis do pliku
            string outGZ = path + ".gz";
            FileStream outfile = new FileStream(outGZ, FileMode.Create);
            GZipStream compressedzipStream = new GZipStream(outfile,
CompressionMode.Compress, true);
            compressedzipStream.Write(o, 0, o.Length);
            compressedzipStream.Close();

            // zwracamy tablicę ne kompresowaną przez GZip
            return o;

        }
```

```
        private byte[] Compress(byte[, ,] rgb, int cW, int cH)
        {
            // Wartości do kwantowania wskazników wavelet'a
            int[] dwDiv = { 48, 32, 16, 16, 24, 24, 1, 1 };
            int[] dwTop = { 24, 32, 24, 24, 24, 24, 32, 32 };
            int SamplerDiv = 2, SamplerTop = 2;
            // Procenty kwantowania Y, cr, cb składników koloru
            int YPerec = 100, crPerec = 85, cbPerec = 85;
            int WVCount = 6; // ilość poziomów splotu falkowego
            // Kodowanie z RGB do YCrCb
            double[, ,] YCrCb = YCrCbEncode(rgb, cW, cH, YPerec, crPerec, cbPerec, cW,
cH);
            // Wykonywame splot falkowy po kolejności do każdego kanalu koloru
            for (int z = 0; z < 3; z++)
            {
                // Skręcamy każdy kanał określoną liczbę raz
                for (int dWave = 0; dWave < WVCount; dWave++)
                {
                    int waveW = Convert.ToInt32(cW / Math.Pow(2, dWave));
                    int waveH = Convert.ToInt32(cH / Math.Pow(2, dWave));
                    if (z == 2)
                    {
                        // Kanal ze składnikiem Y kwantujemy do mniejszej wartości,
                        // bowiem on miejsci w sobie strukturę obrazu (składnik jaskrawości), a w
innych kanałach informacja o kolorach
                        YCrCb = WaveletePack(YCrCb, z, waveW, waveH, dwDiv[dWave],
dwTop[dWave], dWave);
                    }
                    else
                    {
                        YCrCb = WaveletePack(YCrCb, z, waveW, waveH, dwDiv[dWave] *
SamplerDiv, dwTop[dWave] * SamplerTop, dWave);
                    }
                }
            }
            // przetwarzanie tablicy do jednowymiarowej
            byte[] flattened = doPack(YCrCb, cW, cH, WVCount);
            return flattened;

        }

        /* Procedura pakuje tablicę liczb typu Double do tablicy typu Byte
        Wskutek obecności duzej liczby wartości, które układają się w granicach bajtu.
        Z początku wszystkie wartości Double są przetwarzane do typu Short.
        Następnie te z wartości, które nie układają się do typu bajt są dopisywane w końcu
wyjściowego strumienia danych, a zamiast ich do tablicy bajtów
        zapisywana jest wartość 255 */
        private byte[] doPack(double[, ,] ImgData, int cW, int cH, int wDepth)
        {
            short Value;
            int lPos = 0;
```

```
int size = cW * cH * 3;
// rezerwowanie dla wartości typu short
int intCount = 0;
short[] shorts = new short[size];
byte[] Ret = new byte[size];
// przejście tablicy po kolejności wg poziomów wejwletów
for (int d = wDepth - 1; d >= 0; d--)
{
    int wSize = (int)Math.Pow(2f, Convert.ToDouble(d));
    int W = cW / wSize;
    int H = cH / wSize;
    int w2 = W / 2;
    int h2 = H / 2;
    // kąt lewy górny
    if (d == wDepth - 1)
    {
        for (int z = 0; z < 3; z++)
        {
            for (int j = 0; j < h2; j++)
            {
                for (int i = 0; i < w2; i++)
                {
                    Value = (short)Math.Round(ImgData[z, i, j]);
                    if ((Value >= -127) && (Value <= 127))
                    {
                        Ret[lPos++] = Convert.ToByte(Value + 127);
                    }
                    else
                    {
                        Ret[lPos++] = 255;
                        shorts[intCount++] = Value;
                    }
                }
            }
        }
    }
    // kąt prawy górny + prawy dolny
    for (int z = 0; z < 3; z++)
    {
        for (int j = 0; j < H; j++)
        {
            for (int i = w2; i < W; i++)
            {
                Value = (short)Math.Round(ImgData[z, i, j]);
                if ((Value >= -127) && (Value <= 127))
                {
                    Ret[lPos++] = Convert.ToByte(Value + 127);
                }
                else
                {
                    Ret[lPos++] = 255;
                    shorts[intCount++] = Value;
```

```
                    }
                }
            }
        }
        // kąt lewy dolny
        for (int z = 0; z < 3; z++)
        {
            for (int j = h2; j < H; j++)
            {
                for (int i = 0; i < w2; i++)
                {
                    Value = (short)Math.Round(ImgData[z, i, j]);
                    if ((Value >= -127) && (Value <= 127))
                    {
                        Ret[lPos++] = Convert.ToByte(Value + 127);
                    }
                    else
                    {
                        Ret[lPos++] = 255;
                        shorts[intCount++] = Value;
                    }
                }
            }
        }
    }
    // połączenie dwóch tablic (byte[] и short[]) do jednej
    int shortArraySize = intCount * 2;
    Array.Resize(ref Ret, Ret.Length + shortArraySize);
    Buffer.BlockCopy(shorts, 0, Ret, Ret.Length - shortArraySize, shortArraySize);
    // zwracamy tablicę płaską jako wynik
    return Ret;
}

private double[, ,] WaveletePack(double[, ,] ImgArray, int Component, int cW, int cH,
int dwDevider, int dwTop, int dwStep)
{
    short Value;
    int cw2 = cW / 2;
    int cH2 = cH / 2;
    // obliczenie wskaźnika kwantowania
    double dbDiv = 1f / dwDevider;
    ImgArray = Wv(ImgArray, cW, cH, Component, WV_TOP_TO_BOTTOM);
    ImgArray = Wv(ImgArray, cH, cW, Component, WV_LEFT_TO_RIGHT);
    // kwantowanie
    for (int j = 0; j < cH; j++)
    {
        for (int i = 0; i < cW; i++)
        {
            if ((i >= cw2) || (j >= cH2))
            {
                Value = (short)Math.Round(ImgArray[Component, i, j]);
                if (Value != 0)
```

```
            {
                int value2 = Value;
                if (value2 < 0) { value2 = -value2; }
                if (value2 < dwTop)
                {
                    ImgArray[Component, i, j] = 0;
                }
                else
                {
                    ImgArray[Component, i, j] = Value * dbDiv;
                }
            }
        }
    }
    return ImgArray;
}

// Szybkie podnoszenie dyskretnej bi-ortogonalnej falki CDF 9/7
private double[, ,] Wv(double[, ,] ImgArray, int n, int dwCh, int Component, int Side)
{

    double a;
    int i, j, n2 = n / 2;
    double[] xWavelet = new double[n];
    double[] tempbank = new double[n];

    for (int dwPos = 0; dwPos < dwCh; dwPos++)
    {
        if (Side == WV_LEFT_TO_RIGHT)
        {
            for (j = 0; j < n; j++)
            {
                xWavelet[j] = ImgArray[Component, dwPos, j];
            }
        }
        else if (Side == WV_TOP_TO_BOTTOM)
        {
            for (i = 0; i < n; i++)
            {
                xWavelet[i] = ImgArray[Component, i, dwPos];
            }
        }

        // Predict 1
        a = -1.586134342f;
        for (i = 1; i < n - 1; i += 2)
        {
            xWavelet[i] += a * (xWavelet[i - 1] + xWavelet[i + 1]);
        }

        xWavelet[n - 1] += 2 * a * xWavelet[n - 2];
```

```
        // Update 1
        a = -0.05298011854f;
        for (i = 2; i < n; i += 2)
        {
            xWavelet[i] += a * (xWavelet[i - 1] + xWavelet[i + 1]);
        }
        xWavelet[0] += 2 * a * xWavelet[1];

        // Predict 2
        a = 0.8829110762f;
        for (i = 1; i < n - 1; i += 2)
        {
            xWavelet[i] += a * (xWavelet[i - 1] + xWavelet[i + 1]);
        }
        xWavelet[n - 1] += 2 * a * xWavelet[n - 2];

        // Update 2
        a = 0.4435068522f;
        for (i = 2; i < n; i += 2)
        {
            xWavelet[i] += a * (xWavelet[i - 1] + xWavelet[i + 1]);
        }
        xWavelet[0] += 2 * a * xWavelet[1];

        // Scale
        a = 1f / 1.149604398f;
        j = 0;

        // mnożymy niepasrzyste przez współczynnik "a"
        // dzielimy parzyste przez współczynnik "a"
        if (Side == WV_LEFT_TO_RIGHT)
        {
            for (i = 0; i < n2; i++)
            {
                ImgArray[Component, dwPos, i] = xWavelet[j++] / a;
                ImgArray[Component, dwPos, n2 + i] = xWavelet[j++] * a;
            }
        }
        else if (Side == WV_TOP_TO_BOTTOM)
        {
            for (i = 0; i < n2; i++)
            {
                ImgArray[Component, i, dwPos] = xWavelet[j++] / a;
                ImgArray[Component, n2 + i, dwPos] = xWavelet[j++] * a;
            }
        }

    }
    return ImgArray;
}
```

```
        // Metoda przetwarzania RGB do YCrCb
        private double[, ,] YCrCbEncode(byte[, ,] BytesRGB, int cW, int cH, double Ydiv,
double Udiv, double Vdiv, int oW, int oH)
        {
            double vr, vg, vb;
            double kr = 0.299, kg = 0.587, kb = 0.114, kr1 = -0.1687, kg1 = 0.3313, kb1 = 0.5,
kr2 = 0.5, kg2 = 0.4187, kb2 = 0.0813;
            Ydiv = Ydiv / 100f;
            Udiv = Udiv / 100f;
            Vdiv = Vdiv / 100f;
            double[, ,] YCrCb = new double[3, cW, cH];
            for (int j = 0; j < oH; j++)
            {
                for (int i = 0; i < oW; i++)
                {
                    vb = (double)BytesRGB[0, i, j];
                    vg = (double)BytesRGB[1, i, j];
                    vr = (double)BytesRGB[2, i, j];
                    YCrCb[2, i, j] = (kr * vr + kg * vg + kb * vb) * Ydiv;
                    YCrCb[1, i, j] = (kr1 * vr - kg1 * vg + kb1 * vb + 128) * Udiv;
                    YCrCb[0, i, j] = (kr2 * vr - kg2 * vg - kb2 * vb + 128) * Udiv;
                }
            }
            return YCrCb;
        }

        private unsafe byte[, ,] BmpToBytes_Unsafe(Bitmap bmp)
        {
            BitmapData bData = bmp.LockBits(new Rectangle(new Point(), bmp.Size),
ImageLockMode.ReadOnly, PixelFormat.Format24bppRgb);
            // number of bytes in the bitmap
            int byteCount = bData.Stride * bmp.Height;
            byte[] bmpBytes = new byte[byteCount];
            Marshal.Copy(bData.Scan0, bmpBytes, 0, byteCount); // Copy the locked bytes
from memory
            // don't forget to unlock the bitmap!!
            bmp.UnlockBits(bData);
            byte[, ,] ret = new byte[3, bmp.Width, bmp.Height];
            for (int z = 0; z < 3; z++)
            {
                for (int i = 0; i < bmp.Width; i++)
                {
                    for (int j = 0; j < bmp.Height; j++)
                    {
                        ret[z, i, j] = bmpBytes[j * bmp.Width * 3 + i * 3 + z];
                    }
                }
            }
            return ret;
        }

    }
```

```
}
```

***Plik wvDecompress.cs***:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace WaveleteCompression
{
    class wvDecompress
    {

        // Константы
        public const int WV_LEFT_TO_RIGHT = 0;
        public const int WV_TOP_TO_BOTTOM = 1;

        public byte[] run(byte[] compressed)
        {
            int z;
            int dwDepth = 6; // liczba poziomów splotu falkowego (im wiecej tym lepiej
kompresja)
            // sztywnie okrześlone wymiary obrazu
            int w = 512;
            int h = 512;
            // wymiary obrazu i współczynnik są dostarczane z nagłówku (header) pliku
skompresowanego
            int[] dwDiv = { 48, 32, 16, 16, 24, 24, 1, 1 }, dwTop = { 24, 32, 24, 24, 24, 24, 32, 32
};
            int SamplerDiv = 2, YPerec = 100, crPerec = 85, cbPerec = 85;

            double[, ,] yuv = doUnPack(compressed, w, h, dwDepth);

            // Dekompresja falki
            for (z = 0; z < 2; z++)
            {
                for (int dWave = dwDepth - 1; dWave >= 0; dWave--)
                {
                    int w2 = Convert.ToInt32(w / Math.Pow(2, dWave));
                    int h2 = Convert.ToInt32(h / Math.Pow(2, dWave));
                    WaveleteUnPack(yuv, z, w2, h2, dwDiv[dWave] * SamplerDiv);
                }
            }
            z = 2;
            for (int dWave = dwDepth - 1; dWave >= 0; dWave--)
            {
                int w2 = Convert.ToInt32(w / Math.Pow(2, dWave));
                int h2 = Convert.ToInt32(h / Math.Pow(2, dWave));
                WaveleteUnPack(yuv, z, w2, h2, dwDiv[dWave]);
            }
            // YCrCb dekodowanie i rozkład obrazu do tabeli płaskiej
            byte[] rgb_flatened = this.YCrCbDecode(yuv, w, h, YPerec, crPerec, cbPerec);
            return rgb_flatened;
```

```
}

// Ta procedura jest odwrotną do procedury DoPack w klasie wvCompress.
// Ona z powrotem przetwarza jego do typu (short)double z typu byte[]
private static double[, ,] doUnPack(byte[] Bytes, int cW, int cH, int dwDepth)
{
    int lPos = 0;
    byte Value;
    int intIndex = 0;
    // wymiary w byte'ach obrazu we wyniku
    int size = cW * cH * 3;
    // tablica czasowa współczynników wynikowych falki zwiniętej
    double[, ,] ImgData = new double[3, cW, cH];

    int shortsLength = Bytes.Length - size;
    short[] shorts = new short[shortsLength / 2];
    Buffer.BlockCopy(Bytes, size, shorts, 0, shortsLength);

    for (int d = dwDepth - 1; d >= 0; d--)
    {
        int wSize = (int)Math.Pow(2, d);
        int W = cW / wSize;
        int H = cH / wSize;
        int w2 = W / 2;
        int h2 = H / 2;
        // lewy górny kąt
        if (d == dwDepth - 1)
        {
            for (int z = 0; z < 3; z++)
            {
                for (int j = 0; j < h2; j++)
                {
                    for (int i = 0; i < w2; i++)
                    {
                        Value = Bytes[lPos++];
                        if (Value == 255)
                        {
                            ImgData[z, i, j] = shorts[intIndex++];
                        }
                        else
                        {
                            ImgData[z, i, j] = Value - 127;
                        }
                    }
                }
            }
        }
        // prawy górny + prawy dolny
        for (int z = 0; z < 3; z++)
        {
            for (int j = 0; j < H; j++)
            {
```

```
                for (int i = w2; i < W; i++)
                {
                    Value = Bytes[lPos++];
                    if (Value == 255)
                    {
                        ImgData[z, i, j] = shorts[intIndex++];
                    }
                    else
                    {
                        ImgData[z, i, j] = Value - 127;
                    }
                }
            }
        }
        // lewy dolny kąt
        for (int z = 0; z < 3; z++)
        {
            for (int j = h2; j < H; j++)
            {
                for (int i = 0; i < w2; i++)
                {
                    Value = Bytes[lPos++];
                    if (Value == 255)
                    {
                        ImgData[z, i, j] = shorts[intIndex++];
                    }
                    else
                    {
                        ImgData[z, i, j] = Value - 127;
                    }
                }
            }
        }
    }
    // zwracamy wynik
    return ImgData;
}

// Funkcja przemiatania falki
private void WaveleteUnPack(double[, ,] ImgArray, int Component, int cW, int cH, int
dwDevider)
{
    int cw2 = cW / 2, ch2 = cH / 2;
    double dbDiv = 1f / dwDevider;
    // dekwantowanie wartości
    for (int i = 0; i < cW; i++)
    {
        for (int j = 0; j < cH; j++)
        {
            if ((i >= cw2) || (j >= ch2))
            {
                if (ImgArray[Component, i, j] != 0)
```

```
                {
                    ImgArray[Component, i, j] /= dbDiv;
                }
            }
        }
    }
    // Przemiatanie falki
    for (int i = 0; i < cW; i++)
    {
        reWv(ref ImgArray, cH, Component, i, WV_LEFT_TO_RIGHT);
    }
    for (int j = 0; j < cH; j++)
    {
        reWv(ref ImgArray, cW, Component, j, WV_TOP_TO_BOTTOM);
    }
}

// Procedura zwrotnego prędkiego podnoszenia dyskretnej bi-ortogonalnej falki CDF
9/7
private void reWv(ref double[, ,] shorts, int n, int z, int dwPos, int Side)
{

    double a;
    double[] xWavelet = new double[n];
    double[] tempbank = new double[n];

    if (Side == WV_LEFT_TO_RIGHT)
    {
        for (int j = 0; j < n; j++)
        {
            xWavelet[j] = shorts[z, dwPos, j];
        }
    }
    else if (Side == WV_TOP_TO_BOTTOM)
    {
        for (int i = 0; i < n; i++)
        {
            xWavelet[i] = shorts[z, i, dwPos];
        }
    }

    for (int i = 0; i < n / 2; i++)
    {
        tempbank[i * 2] = xWavelet[i];
        tempbank[i * 2 + 1] = xWavelet[i + n / 2];
    }
    for (int i = 0; i < n; i++)
    {
        xWavelet[i] = tempbank[i];
    }

    // Undo scale
```

```
a = 1.149604398f;
for (int i = 0; i < n; i++)
{
    if (i % 2 != 0)
    {
        xWavelet[i] = xWavelet[i] * a;
    }
    else
    {
        xWavelet[i] = xWavelet[i] / a;
    }
}

// Undo update 2
a = -0.4435068522f;
for (int i = 2; i < n; i += 2)
{
    xWavelet[i] = xWavelet[i] + a * (xWavelet[i - 1] + xWavelet[i + 1]);
}
xWavelet[0] = xWavelet[0] + 2 * a * xWavelet[1];

// Undo predict 2
a = -0.8829110762f;
for (int i = 1; i < n - 1; i += 2)
{
    xWavelet[i] = xWavelet[i] + a * (xWavelet[i - 1] + xWavelet[i + 1]);
}
xWavelet[n - 1] = xWavelet[n - 1] + 2 * a * xWavelet[n - 2];

// Undo update 1
a = 0.05298011854f;
for (int i = 2; i < n; i += 2)
{
    xWavelet[i] = xWavelet[i] + a * (xWavelet[i - 1] + xWavelet[i + 1]);
}
xWavelet[0] = xWavelet[0] + 2 * a * xWavelet[1];

// Undo predict 1
a = 1.586134342f;
for (int i = 1; i < n - 1; i += 2)
{
    xWavelet[i] = xWavelet[i] + a * (xWavelet[i - 1] + xWavelet[i + 1]);
}
xWavelet[n - 1] = xWavelet[n - 1] + 2 * a * xWavelet[n - 2];

if (Side == WV_LEFT_TO_RIGHT)
{
    for (int j = 0; j < n; j++)
    {
        shorts[z, dwPos, j] = xWavelet[j];
    }
}
```

```
            else if (Side == WV_TOP_TO_BOTTOM)
            {
                for (int i = 0; i < n; i++)
                {
                    shorts[z, i, dwPos] = xWavelet[i];
                }
            }
        }

        // Metod przekodowania z YCrCb do RGB
        private byte[] YCrCbDecode(double[, ,] yuv, int w, int h, double Ydiv, double Udiv,
double Vdiv)
        {
            byte[] bytes_flat = new byte[3 * w * h];
            double vr, vg, vb;
            double vY, vCb, vCr;
            Ydiv = Ydiv / 100f;
            Udiv = Udiv / 100f;
            Vdiv = Vdiv / 100f;
            for (int j = 0; j < h; j++)
            {
                for (int i = 0; i < w; i++)
                {
                    vCr = yuv[0, i, j] / Vdiv;
                    vCb = yuv[1, i, j] / Udiv;
                    vY = yuv[2, i, j] / Ydiv;
                    vr = vY + 1.402f * (vCr - 128f);
                    vg = vY - 0.34414f * (vCb - 128f) - 0.71414f * (vCr - 128f);
                    vb = vY + 1.722f * (vCb - 128f);
                    if (vr > 255) { vr = 255; }
                    if (vg > 255) { vg = 255; }
                    if (vb > 255) { vb = 255; }
                    if (vr < 0) { vr = 0; }
                    if (vg < 0) { vg = 0; }
                    if (vb < 0) { vb = 0; }
                    bytes_flat[j * w * 3 + i * 3 + 0] = (byte)vb;
                    bytes_flat[j * w * 3 + i * 3 + 1] = (byte)vg;
                    bytes_flat[j * w * 3 + i * 3 + 2] = (byte)vr;
                }
            }
            return bytes_flat;
        }

    }
}
```

## 3. Zadanie do pracy samodzielnej

3.1. Wykonaj implementację kodu programowego w języku C# dla kompresji i dekompresji obrazów wizualnych w aplikacji **WindowsForms** z menu dialogowym dla określenia adresu pliku obrazu wizualnego. Z początku z plikiem przykładowym, a dalej z plikiem dowolnej formy i wymiarów (pobrać plik z Internet lub wykonać zdjęcie smartfonem). Program ma określić wymiary obrazu w pikseli. Dla pliku, który nie ma formy kwadratowej, lub wymiary nie równają się potęgi liczby 2 program ma dopełnić marginesy w kolorze szarym (RGB = 128,128,128) do formy kwadratowej i wymiarów najbliższych potęgi liczby 2 (na przykład 512×512, 1024×1024 itp.). Sprawdź działanie programu z plikiem przykładowym i z obranym samodzielnym.

3.2. Zdefiniuj pojęcie falki i algorytmu kompresji i dekompresji obrazów.

3.3. Kody i pliki z obrazami zapakuj do jednego pliku archiwum i wrzuć do personalnego konta na Delcie