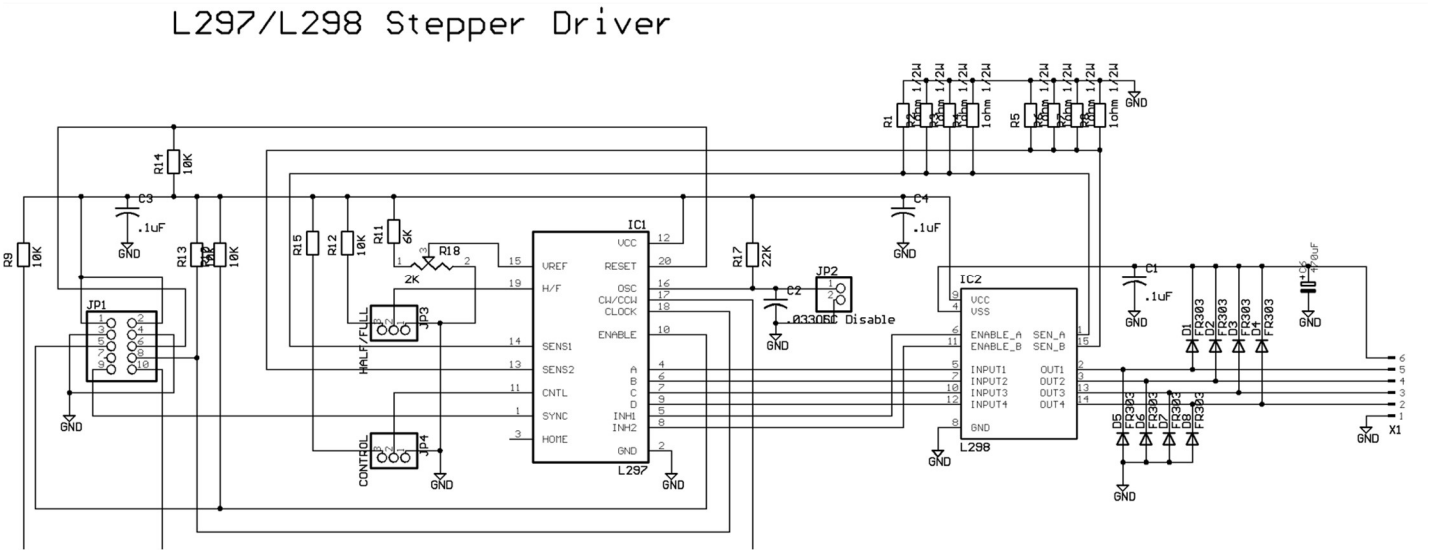
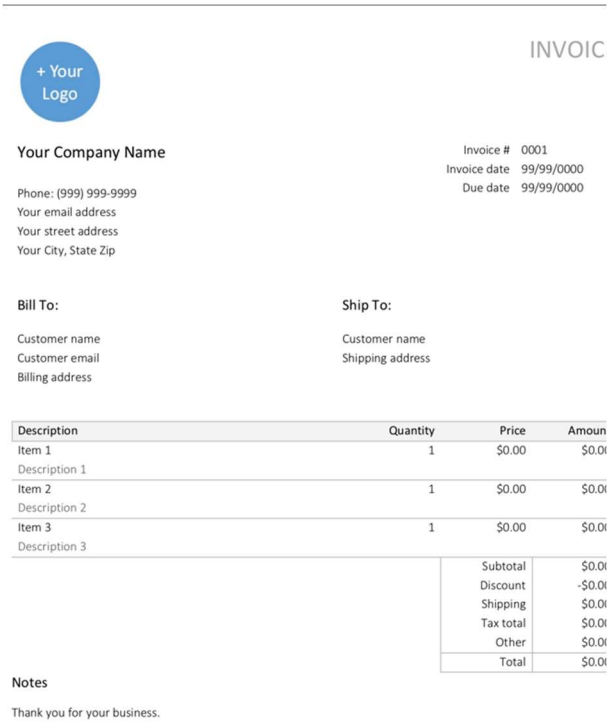
SM LAB_3 Sprawozdanie
Lewicki Maciej

1. Znaleźć 3 obrazy (Rozmiar minimalny 800x600 mogą być większe.) do przeprowadzanie analizy skuteczności kompresji, każdy ma reprezentować jedną z kategorii (0.1 pkt):
    1. Rysunek techniczny,
    2. Skan dokumentu,
    3. Kolorowe zdjęcie.





+ Your Logo

INVOIC

Your Company Name

Phone: (999) 999-9999
Your email address
Your street address
Your City, State Zip

Invoice # 0001
Invoice date 99/99/0000
Due date 99/99/0000

Bill To:

Customer name
Customer email
Billing address

Ship To:

Customer name
Shipping address

Description	Quantity	Price	Amoun
Item 1 Description 1	1	$0.00	$0.0
Item 2 Description 2	1	$0.00	$0.0
Item 3 Description 3	1	$0.00	$0.0

Subtotal	$0.0
Discount	-$0.0
Shipping	$0.0
Tax total	$0.0
Other	$0.0
Total	$0.0

Notes

Thank you for your business.

2. Napisać kod dla dwóch rodzajów kompresji w formie koder i dekoder. Koder powinien zwracać pojedyncza zmienną, która zawiera również informację o oryginalnym rozmiarze kompresowanej informacji. Dekoder powinien przyjmować tą zmienną i zwracać oryginalną informację. Każdy kod, który będzie to realizował poza funkcją dekodującą lub przekazywał dodatkowe informacje, będzie oceniany negatywnie.
    1. Kompresja RLE. Dla dowolnego rodzaju danych. Można założyć że przekazujemy je w formie tablicy numpy. (0.3 pkt)

Kompresja RLE:

```
def encodeRLE(_data):
    data = _data.copy()
    shape = data.shape
    data = data.flatten()

    # preallocate and encode shape at the start
    newData = np.empty(data.shape[0] * 2 + len(shape) + 1).astype(_data.dtype)
    newData[0] = len(shape)
    shapeIndex = 0
    for shp in shape:
        newData[1 + shapeIndex] = shp
        shapeIndex += 1

    newDataIndex = len(shape) + 1
    currentIndex = 0
    while currentIndex < data.shape[0]:
        current_bit = data[currentIndex]
        n_repeats = 1
        while currentIndex + n_repeats < data.shape[0] and data[currentIndex + n_repeats] == current_bit:
            n_repeats += 1

        currentIndex += n_repeats
        newData[newDataIndex] = n_repeats
        newData[newDataIndex + 1] = current_bit
        newDataIndex += 2

    newData = newData[:newDataIndex].copy()
    return newData
```

Dekompresja RLE:

```
def decodeRLE(data):
    shpCount = data[0].astype(int)
    shape = np.empty(shpCount)

    size = 1
    for i in range(0, shpCount):
        shape[i] = data[i + 1]
        size *= shape[i]
    shape = tuple(shape.astype(int))

    newData = np.empty(int(size)).astype(data.dtype)
    currentIndex = shpCount + 1
    newDataIndex = 0
    while currentIndex < data.shape[0]:
        n_repeats = data[currentIndex]
        for i in range(0, n_repeats.astype(int)):
            newData[newDataIndex] = data[currentIndex + 1]
            newDataIndex += 1
        currentIndex += 2

    newData = np.reshape(newData, shape).astype(data.dtype)  # numpy keeps changing the array type on me
    return newData
```

2. Kompresja Quad Tree - Drzewo czwórkowe. Zakładamy, że na wejściu dostajemy obraz kolorowy lub w skali odcieni szarości. Kodujemy wszystkie warstwy na raz, więc kodowanie jest niezależnie od ich ilości, tylko sam kolor będzie miał tyle wartości ile jest warstw. (0.4 pkt)

Kompresja QuadTree:

```
def createQuadTree(img, maxLevel=-1, level=0):
    global nodeCount
    nodeCount += 1
    color = np.mean(img, axis=(0, 1)).astype(int)
    final = (img == color).all()
    resolution = np.array([img.shape[0], img.shape[1]]).astype(int)
    topleft = None
    topright = None
    bottomleft = None
    bottomright = None

    if not final and level != maxLevel:
        if img.shape[0] > 1 and img.shape[1] > 1:
            split_h = np.array_split(img, 2, axis=0)
            split_top = np.array_split(split_h[0], 2, axis=1)
            split_bottom = np.array_split(split_h[1], 2, axis=1)
            topleft =     createQuadTree(split_top[0], maxLevel, level + 1)
            topright =    createQuadTree(split_top[1], maxLevel, level + 1)
            bottomleft =  createQuadTree(split_bottom[0], maxLevel, level + 1)
            bottomright = createQuadTree(split_bottom[1], maxLevel, level + 1)
        elif img.shape[0] == 1:
            split = np.array_split(img, 2, axis=1)
            topleft = createQuadTree(split[0], maxLevel, level + 1)
            topright = createQuadTree(split[1], maxLevel, level + 1)
        elif img.shape[1] == 1:
            split = np.array_split(img, 2, axis=0)
            topleft = createQuadTree(split[0], maxLevel, level + 1)
            bottomleft = createQuadTree(split[1], maxLevel, level + 1)

    return Node(color, final, level, resolution, topleft, topright, bottomleft, bottomright)
```

Dekompresja QuadTree:

```python
def joinQuadrants(top_left, top_right, bottom_left, bottom_right):
    if top_right is None:
        return np.concatenate((top_left, bottom_left), axis=0)
    elif bottom_left is None:
        return np.concatenate((top_left, top_right), axis=1)

    top = np.concatenate((top_left, top_right), axis=1)
    bottom = np.concatenate((bottom_left, bottom_right), axis=1)
    return np.concatenate((top, bottom))


def getImageFromTree(node, level=-1):
    if node is None:
        return None
    if node.final or node.level == level:
        return np.tile(node.color, (node.resolution[0], node.resolution[1], 1))
    else:
        return joinQuadrants(getImageFromTree(node.topleft, level), getImageFromTree(node.topright, level),
                             getImageFromTree(node.bottomleft, level), getImageFromTree(node.bottomright, level)).astype(int)
```

3. Krótkie sprawozdanie/raport z prac (0.1 pkt)
    1. Proszę poświęcić paragraf sprawozdania i opisać sposób wykorzystania pamięci przez wasze implementacje. Gdzie przechowywanych jest rozmiar obrazu w każdym z skompresowanych plików oraz jaką strukturą w pamięci jest drzewo.

RLE:

Kompresja RLE odbywa się przy użyciu zwykłego ndarray, do którego wpierw wprowadzane są informację dotyczące rozmiaru zdjęcia a następnie wszystkie skompresowane dane.

QuadTree:

QuadTree jest zrealizowane obiektowo na bazie „linked list”, w każdym „node” jest przechowywane: poziom, 4 wskaźniki na dzieci, rozdzielczość, średni kolor oraz to czy jest to liść. Po kompresji zachowujemy wszystkie poziomy dzięki czemu struktura pozwala nam obejrzeć obraz w różnych poziomach „ostrości” powoduje to jednak, że poziom kompresji dla wielu przykładów będzie poniżej 1, tj. kompresja będzie negatywna.

Oczywiście można napisać dodatkową metodę która zapisze wszystkie końcowe „node” lub wszystkie „node” na określonym poziomie w innej strukturze i odrzuci resztę, co poprawiłoby kompresję wielokrotnie. Przechowywanie całego drzewa ma sens jedynie gdy chcemy wykorzystać różne poziomy rozdzielczości.

2. Dla każdego z przygotowanych obrazów testowych przeprowadzić kompresję i dekompresję, każda z metod. Udowodnić poprawność (identyczność) danych po dekompresji z danymi źródłowymi. Sprawdzić skuteczność kompresji - zarówno jako jej stopień jak i jaki % oryginalnego obrazu stanowi obraz po kompresji.

Dowód poprawności danych zapewnia asercja przy wykonywaniu testów:

```
print('\n\n------------RLE------------')
print('Original size:     ', origSize, '   pixelCount: ', image.shape[0]*image.shape[1])
compressed = encodeRLE(image)
rleSize = get_size(compressed)
print('Compressed size:   ', rleSize)
decompressed = decodeRLE(compressed)
assert (image == decompressed).all
print('Decompressed size: ', get_size(decompressed))
print('Compression level: ', round(origSize / rleSize, 2))
print('Compression pct:   ', round(100 * rleSize / origSize, 2))
```

Obraz „schemat.png”

```
------------RLE------------
Original size:      28345248    pixelCount:  2362104
Compressed size:    399112
Decompressed size:  28345248
Compression level:  71.02
Compression pct:    1.41


---------QuadTree----------
Original size:      28345248    pixelCount:  2362104
Compressed size:    16986814    nodeCount:   235925    avgNodeSize:  72
Decompressed size:  28345248
Compression level:  1.67
Compression pct:    59.93
```

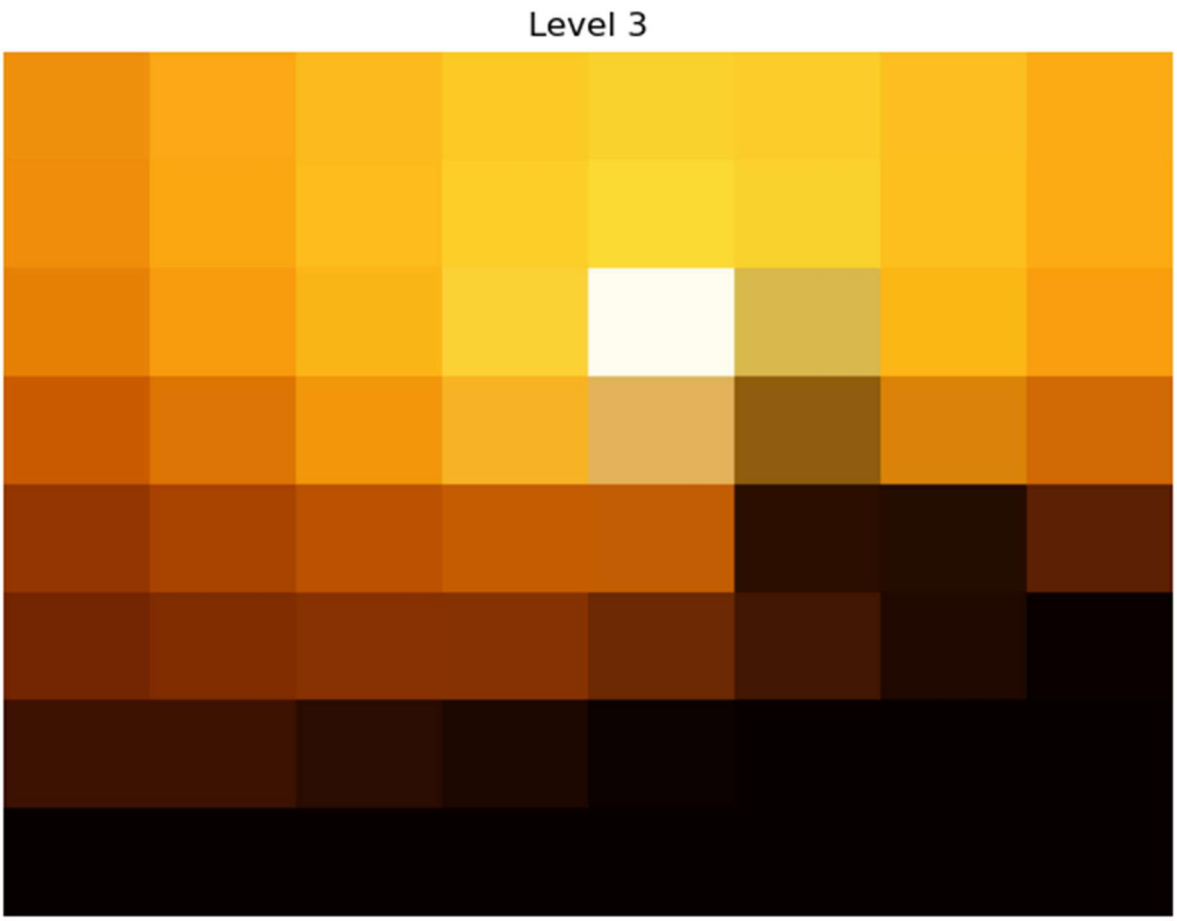
Obraz „scan.jpg”

```
------------RLE------------
Original size:      25245000    pixelCount:  2103750
Compressed size:    1407928
Decompressed size:  25245000
Compression level:  17.93
Compression pct:    5.58


---------QuadTree----------
Original size:      25245000    pixelCount:  2103750
Compressed size:    19353728    nodeCount:   268799    avgNodeSize:  72
Decompressed size:  25245000
Compression level:  1.3
Compression pct:    76.66
```

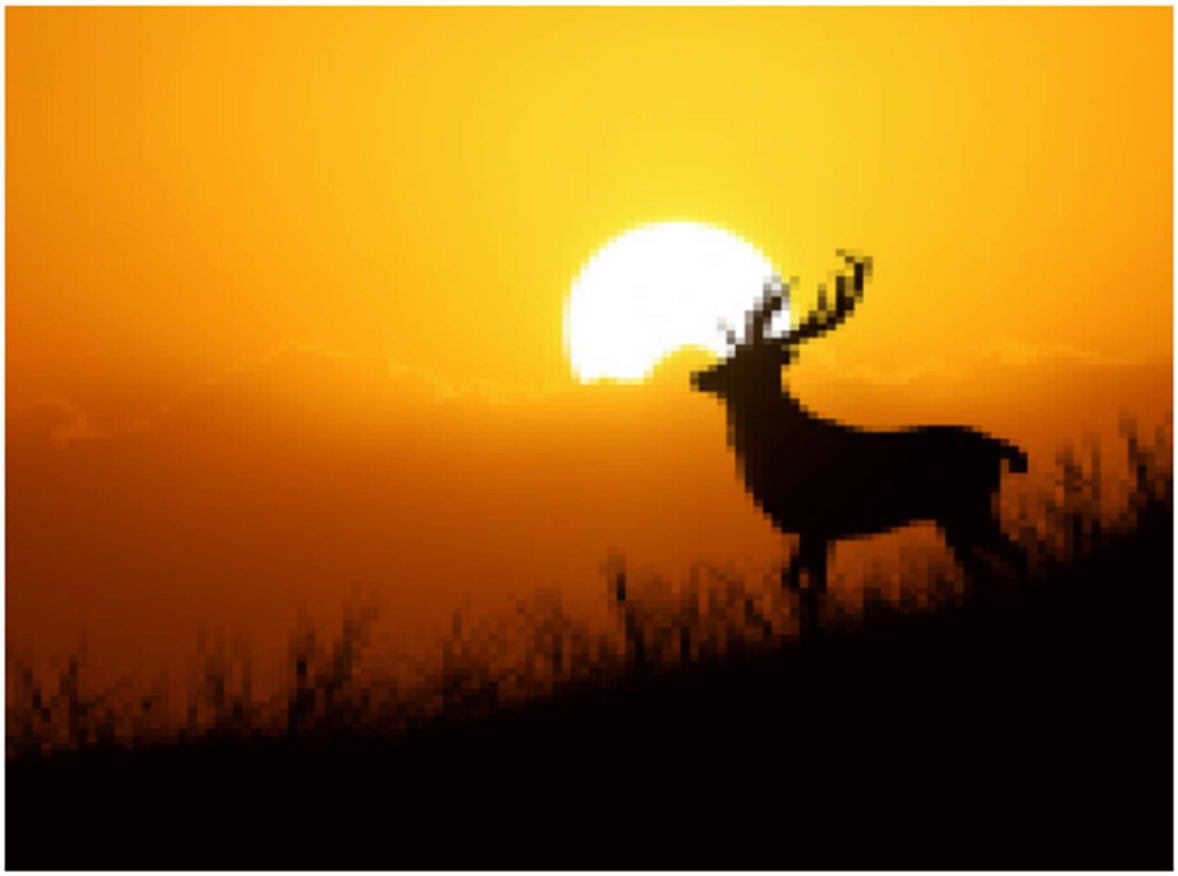
Obraz „photo.jpg”

```
------------RLE------------
Original size:      22732800    pixelCount:  1894400
Compressed size:    40938880
Decompressed size:  22732800
Compression level:  0.56
Compression pct:    180.09


---------QuadTree----------
Original size:      22732800    pixelCount:  1894400
Compressed size:    133085072    nodeCount:   1848401    avgNodeSize:  72
Decompressed size:  22732800
Compression level:  0.17
Compression pct:    585.43
```

Demonstracja możliwości wyboru „poziomu” w QuadTree:



Level 7



Level Maximum