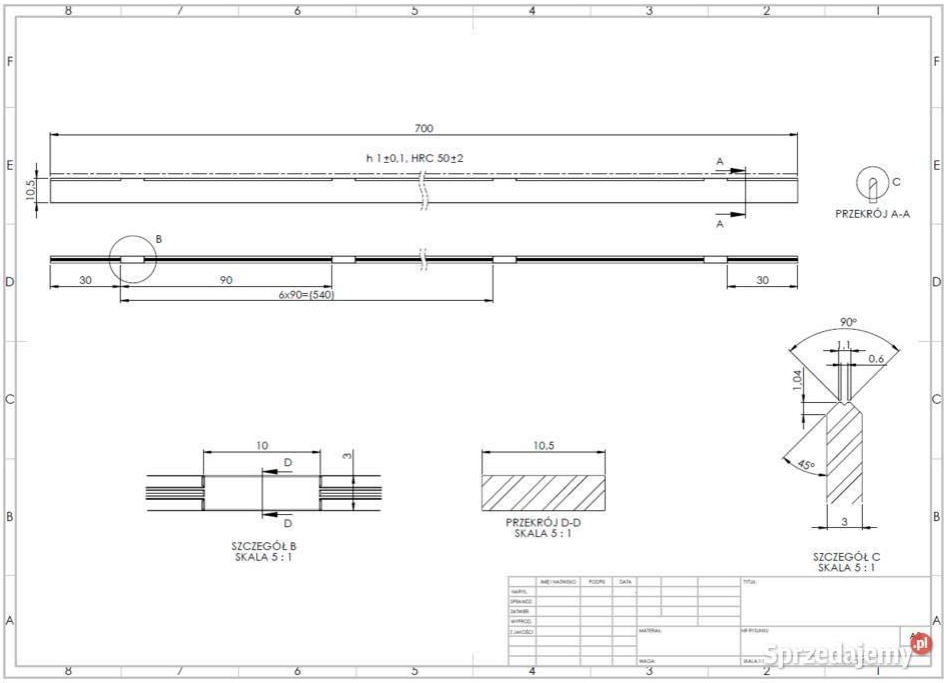
Systemy multimedialne

Daniel Brzezicki (bd46477)

Lab – 18.04.2022

Zadanie 1

1. Znaleźć 3 obrazy do przeprowadzanie analizy skuteczności kompresji (Rozmiar minimalny 800x600 mogą być większe.), każdy ma reprezentować jedną z kategorii (0,1 pkt):



Na zajęciach edukacji naukowo – informatycznej nauczę się:

- gdzie szukać dokumentów,
- pisania referatów naukowych,
- poruszania się po świecie bibliografii,
- korzystania z baz danych w Internecie, itp.

DOKUMENT – myśl ludzka utrwalona na nośniku materialnym i przeznaczona do rozpowszechniania.

Typy dokumentów:

1 pierwotne – dokument w postaci nadanej mu bezpośrednio przez autora lub wydawcę (książka),
2 pochodne – dokumenty, które zostały opracowane na podstawie dokumentu pierwotnego. Zawierają jakąś dodatkową informację, np. charakterystykę treści, słowo o autorze, liczba stron, wydawnictwo itp.,
3 wtórne – dokument, który jest reprodukcją dokumentu pierwotnego lub pochodny utrwalony na innym nośniku, np. mikrofilm, zdjęcia, skan.

WYDANIE – opublikowanie zespołu egzemplarzy dokumentów wyprodukowanych z jednego wzorca. Wydanie ma swoją określoną czasoprzestrzeń.

WZNOWIENIE – reedycja, kolejne wydanie, opublikowanie dzieła, może zawierać zmiany lub wyglądać jak pierwotne, nie zmienione.

Podział dokumentów ze względu na:

I nośnik;
II formę zapisu treści:
1 dokument wizualny – przeznaczony do oglądania, zawiera utrwalony zapis obrazu (niemy film, slajdy, przezrocza),
2 dokument dźwiękowy (audialny) – do słuchania, np. płyty Audio-CD,
3 dokument graficzny – dokument pisany, treść jest wyrażona za pomocą znaków graficznych, np. pisma,
4 dokument ikonograficzny – dokument graficzny przedstawiony za pomocą obrazu,
5 dokument audiowizualny – np. film,
6 dokument multimedialny – np. słowniki, encyklopedie multimedialne, obraz, dźwięk i pismo połączone ze sobą za pomocą ścieżek,
7 dokument kartograficzny – zapis treści w formie mapy albo w formie kuli, np. globusy, atlasy.
III punktu widzenia zamierzenia wydawniczego:
1 dokument niesamoistny wydawniczo – sam nie może istnieć, stanowi większą część wydawnictwa (np. artykuł w czasopiśmie).
2 dokument samoistny wydawniczo – może istnieć sam jako oddzielna, zamknięta całość dokumentu zwartego, wydawnictwa ciągłego (np. książka, czasopismo, wolumin).

Zadanie 2

2. Kompresja strumieniową (RLE lub ByteRun). Powinna działać dla dowolnego rozmiaru danych (np. macierz 22 wymiarowa). Zakładamy że przekazujemy do środka jako tablicę w formacie numpy. (0,35 pkt)
3. 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. (0,45 pkt)

```python
def RLE(data):
    d = data.copy()
    d = d.flatten()

    newData = []

    newData.append(len(data.shape))
    for i in range(len(data.shape)):
        newData.append(data.shape[i])

    counter=0

    current = d[0]
    for i in range(d.shape[0]):
        if(d[i]==current):
            counter+=1
        else:
            newData.append(counter)
            newData.append(current)
            current = d[i]
            counter=1

    newData.append(counter)
    newData.append(current)


    return newData
```

```python
def decodeRLE(data):
    newData=[]
    counters=[]
    values=[]

    shape=[]
    size = data[0]
    data.pop(0)
    for i in range(size):
        shape.append(data[0])
        data.pop(0)

    for i in range(len(data)):
        if(i%2==0):
            counters.append(data[i])
        else:
            values.append(data[i])

    value = 0
    for i in counters:
        for j in range(i):
            newData.append(values[value])
        value+=1


    newData = np.array(newData).reshape(tuple(shape[i] for i in range(len(shape))))
    return newData
```

```python
def quadOper(source, x, w, y, h):
    tree = QuadTree(0,x, w, y, h)
    src = source[x:w, y:h]

    leafPos = check(src,0,src[0])
    if(leafPos):
        leafPos= check(src,1,src[:,0])

    if leafPos:
        tree.setLeaf(source[x,y])
    else:
        if (h - y) == 1:
            tree.addNode(quadOper(source, x, (w + x) // 2, y, h))
            tree.addNode(quadOper(source, ((w + x) // 2), w, y, h))
        elif (w - x) == 1:
            tree.addNode(quadOper(source, x, w, y, (y + h) // 2 ))
            tree.addNode(quadOper(source, x, w, ((y + h) // 2 ), h))
        else:
            tree.addNode(quadOper(source, x, (w + x) // 2, y, (y + h) // 2 ))
            tree.addNode(quadOper(source, ((w + x) // 2), w, ((y + h) // 2 ), h))
            tree.addNode(quadOper(source, ((w + x) // 2), w, y, (y + h) // 2 ))
            tree.addNode(quadOper(source, x, (w + x) // 2, ((y + h) // 2 ), h))

    return tree
```

```python
def quadEncode(source):
    return quadOper(source, 0, source.shape[0], 0, source.shape[1])

def decode(tree,decoded):
    nCount = tree.getNodesCount()
    if nCount > 0:
        for i in range(nCount):
            decoded = decode(tree.nodes[i], decoded)
    else:
        decoded[tree.x:tree.width, tree.y:tree.height] = tree.leaf
    return decoded

def quadDecode(tree):
    result = np.zeros((tree.width, tree.height, 3)).astype(int)
    result = decode(tree, result)

    return result
```

Zadanie 3

Kompresja RLE została zaimplementowana przy użyciu “pythonowskich” list. Pierw wprowadzane są dane o kompresowanym zdjęciu, następnie zaimplementowana jest sama kompresja. Poprawność danych jest sprawdzana przy pomocy porównania tablic.

Drzewo czwórkowe zostało zaimplementowane przy użyciu listy node’ów, gdzie przechowywane są kolejne (potomki) obiekty klasy QuadTree. Klasa QuadTree zawiera listę node’ów, leaf, x, y, width, height. Całość operacji tworzenia drzewa zachodzi w funkcji quadOper i funkcji pomocnej check. Poprawność sprawdzana identycznie jak w kompresji RLE wyżej.

Identyczność

```python
def ShowResults(originalData, compressedData, decompressedData, title):
    originalSize= get_size(originalData)
    compressedSize = get_size(compressedData)
    decompressedSize = get_size(decompressedData)

    compare = originalData==decompressedData

    print(title)
    print("Wielkosc zdekompresowana: ",decompressedSize)
    print("Stopien kompresji: ", originalSize/compressedSize)
    print("Wielkosc w procentach: ",100*compressedSize/originalSize," %")
    print("Czy dane sa identyczne jak oryginal?: ", compare.all())
    plt.imshow(decompressedData)
    plt.show()
```

Skan dokumentu:

```
======= RLE:
Wielkosc zdekompresowana:  6311250
Stopien kompresji:  0.4228725098931564
Wielkosc w procentach:  236.4778926520103  %
Czy dane sa identyczne jak oryginal?:  True
======= Drzewo czworkowe:
Wielkosc zdekompresowana:  25245000
Stopien kompresji:  0.029999679575614942
Wielkosc w procentach:  3333.368936423054  %
Czy dane sa identyczne jak oryginal?:  True
```

Zdjecia kolorowe (żyrafa):

```
======= RLE:
Wielkosc zdekompresowana:  3240000
Stopien kompresji:  0.02392976702008372
Wielkosc w procentach:  4178.895679012346  %
Czy dane sa identyczne jak oryginal?:  True
======= Drzewo czworkowe:
Wielkosc zdekompresowana:  12960000
Stopien kompresji:  0.007007824536739594
Wielkosc w procentach:  14269.763672839506  %
Czy dane sa identyczne jak oryginal?:  True
```

Rysunek techniczny:

```
======= RLE:
Wielkosc zdekompresowana:  1995840
Stopien kompresji:  0.4345225822899565
Wielkosc w procentach:  230.13763628346962  %
Czy dane sa identyczne jak oryginal?:  True
======= Drzewo czworkowe:
Wielkosc zdekompresowana:  7983360
Stopien kompresji:  0.028240319652843748
Wielkosc w procentach:  3541.036405723906  %
Czy dane sa identyczne jak oryginal?:  True
```