

Kod rozwiązania:

RLE:

```
def rle_encode(arr): 3 usages
    # Flatten the array and get its shape
    flat_arr = arr.flatten()
    shape = arr.shape

    # Initialize the encoded data list
    encoded_data = []

    # Perform RLE encoding with progress bar
    prev_value = flat_arr[0]
    count = 1
    for value in tqdm(flat_arr[1:], desc="Encoding"):
        if value == prev_value:
            count += 1
        else:
            encoded_data.append(prev_value)
            encoded_data.append(count)
            prev_value = value
            count = 1
    encoded_data.append(prev_value)
    encoded_data.append(count)

    # Encode the shape at the beginning
    shape_encoded = np.array([len(shape)] + list(shape))
    encoded_data = np.concatenate([shape_encoded, encoded_data])

    return encoded_data

def rle_decode(encoded_data): 3 usages
    # Extract the shape information
    shape_length = int(encoded_data[0])
    shape = tuple(encoded_data[1:1 + shape_length].astype(int))
    encoded_data = encoded_data[1 + shape_length:]

    # Initialize the decoded array
    decoded_arr = []

    # Perform RLE decoding with progress bar
    for i in tqdm(range(0, len(encoded_data), 2), desc="Decoding"):
        value = encoded_data[i]
        count = encoded_data[i + 1]
        decoded_arr.extend([value] * count)

    # Convert the list back to a NumPy array and reshape it
    decoded_arr = np.array(decoded_arr).reshape(shape)

    return decoded_arr
```

ByteRun:

```
def byterun_encode(arr): 3 usages
    # Flatten the array and get its shape
    flat_arr = arr.flatten()
    shape = arr.shape

    # Initialize the encoded data list
    encoded_data = []

    # Perform byte run-length encoding with progress bar
    prev_value = flat_arr[0]
    count = 1
    for value in tqdm(flat_arr[1:], desc="ByteRun Encoding"):
        if value == prev_value and count < 255:
            count += 1
        else:
            encoded_data.append(prev_value)
            encoded_data.append(count)
            prev_value = value
            count = 1
    encoded_data.append(prev_value)
    encoded_data.append(count)

    # Encode the shape at the beginning
    shape_encoded = np.array([len(shape)] + list(shape))
    encoded_data = np.concatenate([shape_encoded, encoded_data])

    return encoded_data

def byterun_decode(encoded_data): 3 usages
    # Extract the shape information
    shape_length = int(encoded_data[0])
    shape = tuple(encoded_data[1:1 + shape_length].astype(int))
    encoded_data = encoded_data[1 + shape_length:]

    # Initialize the decoded array
    decoded_arr = []

    # Perform byte run-length decoding with progress bar
    for i in tqdm(range(0, len(encoded_data), 2), desc="ByteRun Decoding"):
        value = encoded_data[i]
        count = encoded_data[i + 1]
        decoded_arr.extend([value] * count)

    # Convert the list back to a NumPy array and reshape it
    decoded_arr = np.array(decoded_arr).reshape(shape)

    return decoded_arr
```

1. Sposób wykorzystania pamięci. Gdzie przechowywany jest rozmiar obrazu.

Wykorzystanie pamięci jest zoptymalizowane poprzez kodowanie kształtu oryginalnej tablicy na początku skompresowanych danych. Pozwala to na odtworzenie oryginalnych wymiarów podczas procesu dekompresji. Konkretnie, kształt tablicy jest przechowywany jako pierwsze elementy skompresowanej tablicy danych.

Dla funkcji *rle_encode* i *byterun_encode*, kształt oryginalnej tablicy jest spłaszczany, a jego wymiary są przechowywane na początku skompresowanej tablicy danych. Jest to realizowane poprzez utworzenie tablicy NumPy, która zawiera długość kształtu, a następnie sam kształt. Informacje o kształcie są następnie łączone z zakodowanymi danymi. Podczas dekompresji, funkcje *rle_decode* i *byterun_decode* wyodrębniają te informacje o kształcie z początku skompresowanej tablicy danych, aby poprawnie przekształcić zdekompresowaną tablicę z powrotem do jej oryginalnych wymiarów.

```
def rle_encode(arr): 3 usages
    # Flatten the array and get its shape
    flat_arr = arr.flatten()
    shape = arr.shape

    # Initialize the encoded data list
    encoded_data = []

    # Perform RLE encoding with progress bar
    prev_value = flat_arr[0]
    count = 1
    for value in tqdm(flat_arr[1:], desc="Encoding"):
        if value == prev_value:
            count += 1
        else:
            encoded_data.append(prev_value)
            encoded_data.append(count)
            prev_value = value
            count = 1
    encoded_data.append(prev_value)
    encoded_data.append(count)

    # Encode the shape at the beginning
    shape_encoded = np.array([len(shape)] + list(shape))
    encoded_data = np.concatenate([shape_encoded, encoded_data])

    return encoded_data

def rle_decode(encoded_data): 3 usages
    # Extract the shape information
    shape_length = int(encoded_data[0])
    shape = tuple(encoded_data[1:1 + shape_length].astype(int))
    encoded_data = encoded_data[1 + shape_length:]

    # Initialize the decoded array
    decoded_arr = []

    # Perform RLE decoding with progress bar
    for i in tqdm(range(0, len(encoded_data), 2), desc="Decoding"):
        value = encoded_data[i]
        count = encoded_data[i + 1]
        decoded_arr.extend([value] * count)

    # Convert the list back to a NumPy array and reshape it
    decoded_arr = np.array(decoded_arr).reshape(shape)

    return decoded_arr
```

Zapis informacji o kształcie w kodowaniu

Odczyt informacji o kształcie w dekodowaniu

2. Sprawdzenie poprawności działania kodowania dla danych testowych.

```
tests = (  
    np.array([1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1]),  
    np.array([1, 2, 3, 1, 2, 3, 1, 2, 3]),  
    np.array([5, 1, 5, 1, 5, 5, 1, 1, 5, 5, 1, 1, 5]),  
    np.array([-1, -1, -1, -5, -5, -3, -4, -2, 1, 2, 2, 1]),  
    np.zeros(shape=(1, 520), dtype=np.int64),  
    np.arange(0, 521, 1),  
    np.eye(N=7, dtype=np.int64),  
    np.dstack([np.eye(N=7, dtype=np.int64), np.eye(N=7, dtype=np.int64), np.eye(N=7, dtype=np.int64)]),  
    np.ones(shape=(1, 1, 1, 1, 1, 1, 10), dtype=np.int64)  
)  
  
def przypadki_testowe_rle():  
    for test in tests:  
        encoded_data = rle_encode(test)  
        decoded_arr = rle_decode(encoded_data)  
  
        print("Original array:")  
        print(test)  
        print("Encoded data:")  
        print(encoded_data)  
        print("Decoded array:")  
        print(decoded_arr)  
        print("Arrays are equal:", np.array_equal(test, decoded_arr))  
        print()  
  
def przypadki_testowe_byterun():  
    for test in tests:  
        encoded_data = byterun_encode(test)  
        decoded_arr = byterun_decode(encoded_data)  
  
        print("Original array:")  
        print(test)  
        print("Encoded data:")  
        print(encoded_data)  
        print("Decoded array:")  
        print(decoded_arr)  
        print("Arrays are equal:", np.array_equal(test, decoded_arr))  
        print()
```

Kod służący do testowania działania kodowania i dekodowania.

```
Encoded data:  
[ 1 14  1  4  2  1  1  4  2  1  1  4]  
Arrays are equal: True  
  
Encoded data:  
[1 9 1 1 2 1 3 1 1 1 2 1 3 1 1 1 2 1 3 1]  
Arrays are equal: True  
  
Encoded data:  
[ 1 13  5  1  1  1  5  1  1  1  5  2  1  2  5  2  1  2  5  1]  
Arrays are equal: True  
  
Encoded data:  
[ 1 12 -1  3 -5  2 -3  1 -4  1 -2  1  1  1  2  2  1  1]  
Arrays are equal: True
```

```
Encoded data:
[ 2  1 520  0 520]
Arrays are equal: True

Encoded data:
[ 1 521  0 ...  1 520  1]
Arrays are equal: True

Encoded data:
[2 7 7 1 1 0 7 1 1 0 7 1 1 0 7 1 1 0 7 1 1 0 7 1 1]
Arrays are equal: True

Encoded data:
[ 3  7  7  3  1  3  0 21  1  3  0 21  1  3  0 21  1  3  0 21  1  3  0 21
 1  3  0 21  1  3]
Arrays are equal: True

Encoded data:
[ 7  1  1  1  1  1  1 10  1 10]
Arrays are equal: True

ByteRun
Encoded data:
[ 1 14  1  4  2  1  1  4  2  1  1  4]
Arrays are equal: True

Encoded data:
[1 9 1 1 2 1 3 1 1 1 2 1 3 1 1 1 2 1 3 1]
Arrays are equal: True

Encoded data:
[ 1 13  5  1  1  1  5  1  1  1  5  2  1  2  5  2  1  2  5  1]
Arrays are equal: True

Encoded data:
[ 1 12 -1  3 -5  2 -3  1 -4  1 -2  1  1  1  2  2  1  1]
Arrays are equal: True

Encoded data:
[ 2  1 520  0 255  0 255  0 10]
Arrays are equal: True

Encoded data:
[ 1 521  0 ...  1 520  1]
Arrays are equal: True

Encoded data:
[2 7 7 1 1 0 7 1 1 0 7 1 1 0 7 1 1 0 7 1 1 0 7 1 1]
Arrays are equal: True

Encoded data:
[ 3  7  7  3  1  3  0 21  1  3  0 21  1  3  0 21  1  3  0 21  1  3  0 21
 1  3  0 21  1  3]
Arrays are equal: True

Encoded data:
[ 7  1  1  1  1  1  1 10  1 10]
Arrays are equal: True
```

Wynik działania testów bez wypisywania zawartości oryginalnej i zdekodowanej.

3. Działanie na obrazach testowych.

```
def test_compression_methods_with_test_images(): 1 usage
    for image in test_images:
        print(f"Testing image: {image}")

        # Load the image
        import PIL.Image as Image
        img = np.array(Image.open(image), dtype=np.int64)

        # RLE
        encoded_rle = rle_encode(img)
        decoded_rle = rle_decode(encoded_rle)
        cr_rle, pr_rle = calculate_compression_metrics(img, encoded_rle)
        print(f"RLE Compression Ratio (CR): {cr_rle:.2f}")
        print(f"RLE Percentage Reduction (PR): {pr_rle:.2f}%")
        print("RLE Arrays are equal:", np.array_equal(img, decoded_rle))
        print()


        # ByteRun
        encoded_byterun = byterun_encode(img)
        decoded_byterun = byterun_decode(encoded_byterun)
        cr_byterun, pr_byterun = calculate_compression_metrics(img, encoded_byterun)
        print(f"ByteRun Compression Ratio (CR): {cr_byterun:.2f}")
        print(f"ByteRun Percentage Reduction (PR): {pr_byterun:.2f}%")
        print("ByteRun Arrays are equal:", np.array_equal(img, decoded_byterun))
        print()
```

Kod służący do testowania kodowania i dekodowania na obrazach.

```
def calculate_compression_metrics(original, encoded):
    original_size = original.nbytes
    encoded_size = encoded.nbytes
    cr = original_size / encoded_size
    # pr = (1 - (encoded_size / original_size)) * 100
    pr = (encoded_size / original_size) * 100
    return cr, pr
```

Kod obliczający metryki.

Obrazy testowe:



Procenty


gr. A str. 1/2


imię i nazwisko


klasa


data

1. Rysunek, na którym zacienowano 25% figury, to rysunek:

A.

B.

C.

D.

2. Staś ma 30 zł, a jego młodszy brat 60% tej kwoty. Razem mają:

A. 36 zł B. 42 zł C. 31,80 zł D. 48 zł

3. W gospodarstwie pana Nowaka jest 30 owiec, 43 krowy, 15 królików, 30 kur i 2 psy. Kury w tym gospodarstwie stanowią:

A. 40% wszystkich zwierząt C. 25% wszystkich zwierząt
B. 12,5% wszystkich zwierząt D. 10% wszystkich zwierząt

4. Chude mleko zawiera 2% tłuszczu. Ile gramów tłuszczu zawiera 300 g chudego mleka?

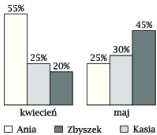
A. 60 g B. 15 g C. 6 g D. 1,5 g

5. Liczba 12,1 to 11% liczby:

A. 6,1 B. 122,2 C. 0,133 D. 110

6. Ania, Zbyszek i Kasia ubiegają się o tytuł najsympatyczniejszego ucznia gimnazjum nr 4. W kwietniu i w maju przeprowadzono ankiety, których wyniki przedstawione są obok. Wybierz zdanie prawdziwe.

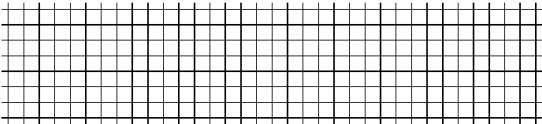
A. Popularność Zbyszka zmniejszyła się o 25 punktów procentowych.
B. Popularność Ani zmniejszyła się o 30 punktów procentowych.
C. Popularność Ani zmniejszyła się o 30%.
D. Popularność Kasi wzrosła o 5%.



| Imię | kwiecień | maj |
|---------|----------|-----|
| Ania | 55% | 25% |
| Zbyszek | 20% | 30% |
| Kasia | 25% | 45% |

7. Zamień podane procenty na ułamki:

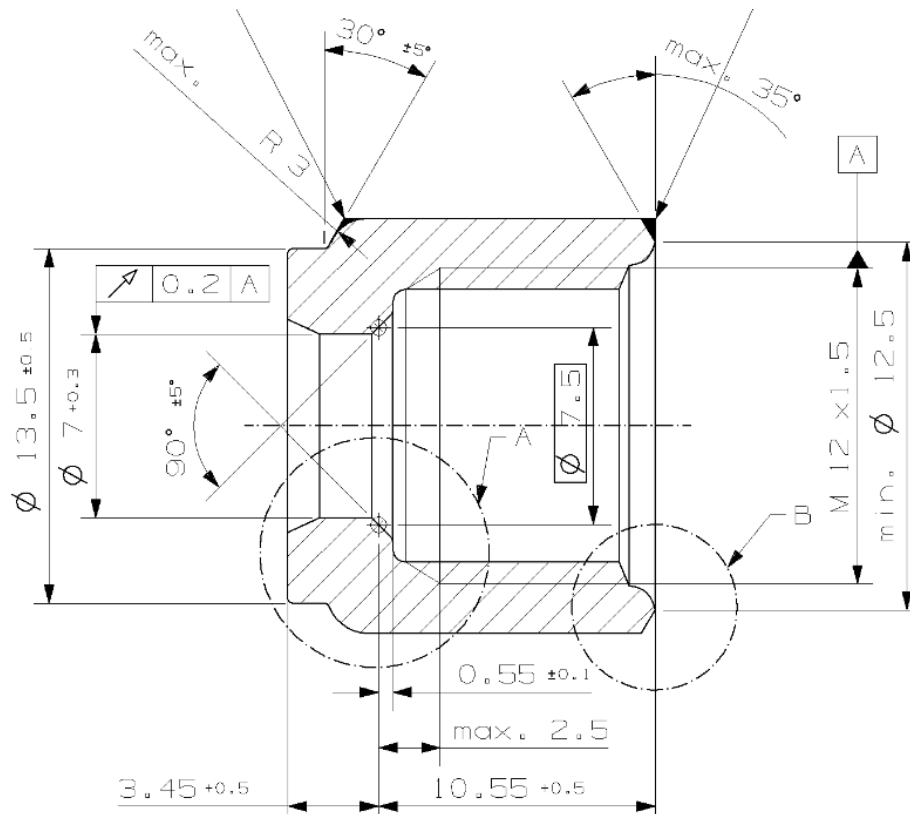
a) 16% b) 4,8% c) 0,72% d) 174%



dokument.png 826 x 1169 px



kolorowy.jpeg 5120 x 2880 px



techniczny.png 1087 x 689 px

Wynik testów:

Testy z metrykami

Testing image: ./images/dokument.png

Encoding: 100%|██████████| 2896781/2896781 [00:00<00:00, 6937458.36it/s]

Decoding: 100%|██████████| 124514/124514 [00:00<00:00, 1627336.05it/s]

RLE Compression Ratio (CR): 11.63

RLE Percentage Reduction (PR): 8.60%

RLE Arrays are equal: True

ByteRun Encoding: 100%|██████████| 2896781/2896781 [00:00<00:00, 6435888.98it/s]

ByteRun Decoding: 100%|██████████| 132696/132696 [00:00<00:00, 1579362.55it/s]

ByteRun Compression Ratio (CR): 10.91

ByteRun Percentage Reduction (PR): 9.16%

ByteRun Arrays are equal: True

Testing image: ./images/kolorowy.jpeg

Encoding: 100%|██████████| 44236799/44236799 [00:08<00:00, 5452006.86it/s]

Decoding: 100%|██████████| 44085379/44085379 [00:14<00:00, 3057142.43it/s]

RLE Compression Ratio (CR): 0.50

RLE Percentage Reduction (PR): 199.32%

RLE Arrays are equal: True

ByteRun Encoding: 100%|██████████| 44236799/44236799 [00:08<00:00, 5502102.83it/s]

ByteRun Decoding: 100%|██████████| 44085379/44085379 [00:14<00:00, 3042243.46it/s]

ByteRun Compression Ratio (CR): 0.50


```
ByteRun Percentage Reduction (PR): 199.32%
ByteRun Arrays are equal: True

Testing image: ./images/techniczny.png
Encoding: 100%|██████████| 3023331/3023331 [00:00<00:00, 6901298.00it/s]
Decoding: 100%|██████████| 54447/54447 [00:00<00:00, 892481.48it/s]
RLE Compression Ratio (CR): 27.76
RLE Percentage Reduction (PR): 3.60%
RLE Arrays are equal: True

ByteRun Encoding: 100%|██████████| 3023331/3023331 [00:00<00:00,
6497777.06it/s]
ByteRun Decoding: 100%|██████████| 62359/62359 [00:00<00:00, 966551.13it/s]
ByteRun Compression Ratio (CR): 24.24
ByteRun Percentage Reduction (PR): 4.13%
ByteRun Arrays are equal: True
```

Podsumowanie

Algorytmy kompresji *RLE* i *ByteRun* działają na zasadzie redukcji ilości danych poprzez zastępowanie powtarzających się wartości i ich licznosci.

W przypadku *RLE*, algorytm przetwarza dane, identyfikując sekwencje powtarzających się wartości i zapisując każdą wartość wraz z liczbą jej powtórzeń. Na przykład, sekwencja [1, 1, 1, 2, 2] zostanie zakodowana jako [1, 3, 2, 2], co oznacza, że wartość 1 powtarza się trzy razy, a wartość 2 dwa razy.

ByteRun działa podobnie, ale dodatkowo ogranicza maksymalną liczbę powtórzeń do 255, co jest przydatne w przypadku danych, gdzie liczba powtórzeń może być bardzo duża.

Oba algorytmy przechowują również informacje o kształcie oryginalnej tablicy na początku skompresowanych danych, co pozwala na dokładne odtworzenie oryginalnych wymiarów podczas dekompresji. Dzięki temu możliwe jest zachowanie struktury danych i ich poprawne odtworzenie po dekompresji.