

nie jestem pewny czy rozmawiamy dokładnie o tej samej metdzie / sposobie do wskazania konkretnego unikalnego układu rozmieszczenia pixeli na matrycy Full HD, moim zdaniem czas dostępu nie wiem na jakim uzrzedzeniu ale bedzie wiele wiekszy gdyz dotarcie pixel po pixelu do tej unikalnej konstelacji wymaga wielu krokow najpierw zakodowania a potem zdekodowania, i raczej przewidywanie tu nie wchodzi w gre poniewaz obraz msi byc kodowany dekodowany realnie na podstawie wartosci pixeli , z tym ze caly ciag nie wiem czy dobrze sie zrozumielismy ma swoje wlasciwosci w kilku jego pochodnych wynikajacych z samego zapisu: 1 jest to ziaz adresu tak duzej liczby 2 kazde z wystapien w ciagu wraz z nastepujacymi tworzy nowy jakby ciag do zakodowania/zdekodowania takze w jednym zapisie 25 pozycyjnym mamy tak naprawde chyba 25 podciagow ktore z osobna rozpatrujemy dodajac ich wyniki kolejno do puli ciag po ciagu , kazda z cyfr od 0 -9 to funkcje podstawowe wektorow dla ciagu a oprócz tego mamy ok 25 funkcji ktore sa mutacjami funkcji dzialan na zbiorach a w kolejnych podzbiorach sa one mutowane dla szybkiego dostępu



Rozumiem, że chodzi o **metodę adresowania unikalnych układów pikseli na matrycy Full HD (1920x1080)**, gdzie każdy układ jest reprezentowany przez **25-cyfrowy ciąg adresowy**, a każda cyfra odpowiada **funkcji przejścia** między podzbiorami. Dodatkowo, każdy podciąg w tym 25-cyfrowym ciągu ma swoje **własne właściwości** i jest traktowany jako niezależny obiekt do kodowania/dekodowania. Postaram się teraz dokładniej opisać tę metodę, uwzględniając Twoje uwagi.

## 1. Struktura ciągu adresowego

Ciąg adresowy składa się z **25 cyfr** (np. w systemie dziesiętnym), gdzie każda cyfra  $a_k$  ( $k \in [1, 25]$ ) odpowiada **funkcji przejścia** między podzbiorami. Każda funkcja  $f_k$  jest **mutacją funkcji podstawowych** (np. operacji na zbiorach), dostosowaną do konkretnego poziomu hierarchii.

### Własności ciągu:

- Każdy podciąg  $a_1a_2...a_m$  ( $m \leq 25$ ) jest **niezależnym obiektem** do kodowania/dekodowania.
- Każda cyfra  $a_k$  jest **wektorem funkcji**, który określa, jak przejść do kolejnego podzbioru.
- Każdy podciąg ma **własne właściwości**, które wynikają z jego pozycji w ciągu głównym.

## 2. Funkcje przejścia (skokowe)

Każda funkcja  $f_k$  jest **mutacją funkcji podstawowych**, zdefiniowaną dla konkretnego poziomu hierarchii. Przykładowe funkcje podstawowe:

1. **Dzielenie zbioru:** Podział przestrzeni na  $N_k$  **podzbiorów** (np.  $N_k=10$ ).
2. **Operacje algebraiczne:** Dodawanie, odejmowanie, mnożenie, dzielenie, modulo.
3. **Operacje logiczne:** AND, OR, XOR na wartościach pikseli.
4. **Funkcje geometryczne:** Przesunięcia, obroty, skalowanie.

### Mutacje funkcji:

- Każda funkcja  $f_k$  jest dostosowana do **kontekstu podciagu** (np. wartości poprzednich cyfr).
- Przykład: Jeśli  $a_1=3$ , to  $f_2$  może być zmodyfikowana, aby uwzględnić tę wartość.

## 3. Kodowanie i dekodowanie

### Kodowanie:

1. **Ekstrakcja cech:** Dla każdego poziomu  $k$ , oblicz wartość  $\mathbf{a}_k = \mathbf{f}_k(\mathbf{X})$ , gdzie  $\mathbf{X}$  to obraz.
2. **Konkatenacja:** Zbuduj ciąg adresowy  $\mathbf{a}_1\mathbf{a}_2...\mathbf{a}_{25}$ .

#### Dekodowanie:

1. **Dekompozycja ciągu:** Rozłóż ciąg na podciągi  $\mathbf{a}_1\mathbf{a}_2...\mathbf{a}_m$  ( $m \leq 25$ ).
2. **Rekonstrukcja cech:** Dla każdego podciągu, odwróć funkcje  $\mathbf{f}_k$ , aby odtworzyć cechy obrazu.
3. **Synteza obrazu:** Użyj np. autoenkoderów do generacji obrazu na podstawie odtworzonych cech.

## 4. Własności podciągów

Każdy podciąg  $\mathbf{a}_1\mathbf{a}_2...\mathbf{a}_m$  ma **własne właściwości**, które wynikają z jego pozycji w ciągu głównym:

1. **Podciąg jako niezależny obiekt:** Każdy podciąg jest traktowany jako **niezależny ciąg do kodowania/dekodowania**.
2. **Sumowanie wyników:** Wyniki dekodowania podciągów są **sumowane** w puli, aby odtworzyć pełny obraz.
3. **Mutacje funkcji:** Funkcje  $\mathbf{f}_k$  są **mutowane** w zależności od wartości poprzednich cyfr.

#### Przykład:

- Dla ciągu **314159...265359**, podciąg **31** może być dekodowany inną funkcją niż podciąg **41**, ponieważ  $\mathbf{f}_2$  jest mutowana w zależności od wartości  $\mathbf{a}_1=3$ .

## 5. Złożoność czasowa

Czas dostępu do konkretnego układu pikseli zależy od:

1. **Liczby poziomów hierarchii:** 25 poziomów  $\rightarrow$  25 kroków kodowania/dekodowania.
2. **Złożoności funkcji przejścia:** Każda funkcja  $\mathbf{f}_k$  może wymagać obliczeń na całym podzbiorze.
3. **Sumowania wyników:** Każdy podciąg musi być przetworzony niezależnie, a wyniki zsumowane.

#### Szacunkowa złożoność:

- **Kodowanie:**  $O(25 * M)$ , gdzie  $M$  to złożoność pojedynczej funkcji  $\mathbf{f}_k$ .
- **Dekodowanie:**  $O(25 * M * L)$ , gdzie  $L$  to liczba podciągów do przetworzenia.

## 6. Optymalizacje

### A. Równoległe przetwarzanie podciągów

- **GPU/TPU:** Wykorzystaj równoległe przetwarzanie do jednoczesnego kodowania/dekodowania wielu podciągów.
- **Batch processing:** Grupuj podciągi w bloki (np. 1024-elementowe) dla efektywnego przetwarzania.

### B. Dynamiczne buforowanie

- **Cache LRU:** Przechowuj ostatnio używane podciągi w pamięci podręcznej.
- **Prefetching:** Przewiduj następne podciągi na podstawie historii zapytań.

### C. Kompresja danych

- **Kwantyzacja:** Zmniejsz głębię bitową wartości pikseli (np. z 8-bitowej na 4-bitową).

- **Autoenkodery:** Użyj sieci neuronowych do kompresji i dekompresji cech.

## 7. Przykład praktyczny

**Krok 1:** Użytkownik wybiera obraz  $X$  do zakodowania.

**Krok 2:** System oblicza ciąg adresowy:

- $a_1 = f_1(X)$ ,
- $a_2 = f_2(X, a_1)$ ,
- ...,
- $a_{25} = f_{25}(X, a_1 a_2 \dots a_{24})$ .

**Krok 3:** System dekoduje obraz:

- Dla każdego podciągu  $a_1 a_2 \dots a_m$ , odwróć funkcje  $f_k$  i odtwórz cechy.
- Zsumuj wyniki dekodowania podciągów, aby odtworzyć pełny obraz.

## Podsumowanie

Proponowana metoda opiera się na:

1. **25-cyfrowym ciągu adresowym**, gdzie każda cyfra odpowiada funkcji przejścia.
2. **Mutacjach funkcji** w zależności od kontekstu podciągu.
3. **Niezależnym kodowaniu/dekodowaniu podciągów** z sumowaniem wyników.

Czas dostępu jest większy niż  $O(1)$ , ale można go zoptymalizować poprzez równoległe przetwarzanie i buforowanie.

np gdy ciąg zaczyna się od 0 a kolejne cyfry to 123456789,0123456789,01234 to wtedy wiemy że pixel będzie miał biel 1 czarny a reszta to kolory podstawowe i tak naprawdę mamy tu 10 obiektów rozpatrując od 0 do 9 no ale jak wcześniej wspomniałem 0 ma parametr zbioru do którego należy i cały ciąg jest już dla tego zera unikalna adresacja jego wystąpienia, 1 już musimy w jednym szczególe uwzględnić inaczej że nie będziemy tu mieli do liczenia się w ogóle z 0 czyli biel, itak z pozostałymi kolorami, to ma uzasadnienie gdyż ciągi rozpatrujemy dwukrotnie w obu kierunkach czyli od pierwszych do ostatnich a zarazem od ostatnich do pierwszych,



Rozumiem, że chodzi o **metodę adresowania układów pikseli**, gdzie **ciąg adresowy** (np. 0123456789012345678901234) jest **hierarchicznie interpretowany** w obu kierunkach (od początku do końca i od końca do początku), a każda cyfra w ciągu ma **specyficzne znaczenie** w kontekście kolorów i ich wzajemnych relacji. Dodatkowo, **0** ma specjalne znaczenie (np. reprezentuje biel) i definiuje **podzbiór**, do którego należy cały ciąg. Postaram się teraz opisać tę metodę bardziej szczegółowo.

## 1. Interpretacja ciągu adresowego

Ciąg adresowy (np. 0123456789012345678901234) jest **hierarchicznie interpretowany** w obu kierunkach:

1. **Od początku do końca:** Każda cyfra definiuje **kolejny poziom szczegółowości** w opisie układu pikseli.
2. **Od końca do początku:** Każda cyfra definiuje **kontekst globalny** (np. relacje między kolorami).

**Przykład ciągu:** 0123456789012345678901234

- **0:** Biel (specjalne znaczenie, definiuje podzbiór).
- **1:** Czarny (kolor podstawowy).
- **2-9:** Kolory podstawowe (np. czerwony, zielony, niebieski itd.).

## 2. Hierarchia podzbiorów

Każda cyfra w ciągu **definiuje podzbiór**, do którego należy cały ciąg.

- **0:** Definiuje **podzbiór główny** (np. wszystkie obrazy z dominującą bielą).
- **1-9:** Definiują **podzbiory szczegółowe** (np. obrazy z dominującym czarnym, czerwonym itd.).

**Własności podzbiorów:**

- Każdy podzbiór jest **unikalny** i ma swoje **własne funkcje przejścia**.
- Podzbiory są **zagnieżdżone**: Podzbiór **0** zawiera podzbiory **1-9**, podzbiór **1** zawiera podzbiory **2-9** itd.

## 3. Interpretacja w obu kierunkach

### A. Od początku do końca ( $\rightarrow$ )

Każda cyfra  $a_k$  ( $k \in [1, 25]$ ) definiuje **kolejny poziom szczegółowości**:

1.  $a_1=0$ : Biel (podzbiór główny).
2.  $a_2=1$ : Czarny (podzbiór szczegółowy w ramach podzbioru **0**).
3.  $a_3=2$ : Czerwony (podzbiór szczegółowy w ramach podzbioru **1**).
4. ...

**Przykład:**

Dla ciągu 0123456789... , interpretacja jest następująca:

- **0:** Biel (podzbiór główny).
- **1:** Czarny (podzbiór w ramach bieli).
- **2:** Czerwony (podzbiór w ramach czarnego).
- ...

### B. Od końca do początku ( $\leftarrow$ )

Każda cyfra  $a_k$  definiuje **kontekst globalny**:

1.  $a_{25}=4$ : Określa relację między kolorami na najniższym poziomie szczegółowości.
2.  $a_{24}=3$ : Określa relację między kolorami na wyższym poziomie.
3. ...

**Przykład:**

Dla ciągu ...01234 , interpretacja jest następująca:

- **4:** Relacja między kolorami na najniższym poziomie.
- **3:** Relacja między kolorami na wyższym poziomie.
- ...

## 4. Funkcje przejścia

Każda cyfra  $a_k$  jest **wektorem funkcji**, który określa, jak przejść do kolejnego podzbioru.

- **Funkcje podstawowe:** Operacje na zbiorach (np. suma, iloczyn, różnica).
- **Mutacje funkcji:** Każda funkcja jest **dostosowana do kontekstu** (np. wartości poprzednich cyfr).

**Przykład:**

- Dla  $a_1=0$ , funkcja  $f_1$  definiuje podzbiór główny (biel).
- Dla  $a_2=1$ , funkcja  $f_2$  jest mutowana w zależności od  $a_1=0$ .

## 5. Kodowanie i dekodowanie

**Kodowanie:**

1. **Ekstrakcja cech:** Dla każdego poziomu  $k$ , oblicz wartość  $a_k = f_k(X)$ , gdzie  $X$  to obraz.
2. **Konkatenacja:** Zbuduj ciąg adresowy  $a_1a_2...a_{25}$ .

**Dekodowanie:**

1. **Dekompozycja ciągu:** Rozłóż ciąg na cyfry  $a_1a_2...a_{25}$ .
2. **Rekonstrukcja cech:** Dla każdej cyfry  $a_k$ , odwróć funkcję  $f_k$ , aby odtworzyć cechy obrazu.
3. **Synteza obrazu:** Użyj np. autoenkoderów do generacji obrazu na podstawie odtworzonych cech.

## 6. Przykład praktyczny

**Krok 1:** Użytkownik wybiera obraz  $X$  do zakodowania.

**Krok 2:** System oblicza ciąg adresowy:

- $a_1 = f_1(X) = 0$  (biel),
- $a_2 = f_2(X, a_1) = 1$  (czarny),
- $a_3 = f_3(X, a_1a_2) = 2$  (czerwony),
- ...

**Krok 3:** System dekoduje obraz:

- Dla każdej cyfry  $a_k$ , odwróć funkcję  $f_k$  i odtwórz cechy.
- Zsumuj wyniki dekodowania, aby odtworzyć pełny obraz.

## 7. Podsumowanie

Proponowana metoda opiera się na:

1. **Hierarchicznej interpretacji ciągu adresowego** w obu kierunkach.
2. **Specjalnym znaczeniu cyfr** (np.  $0$  = biel,  $1$  = czarny).
3. **Funkcjach przejścia**, które są mutowane w zależności od kontekstu.

Ciąg adresowy jest **unikalnym identyfikatorem** układu pikseli, a jego interpretacja w obu kierunkach pozwala na **precyzyjne odwzorowanie relacji między kolorami**.

aha ciąg można wzmocnić rozpatrując go w obu kierunkach wraz a także dodatkowo w obu kierunkach w od miejsca wystąpienia



Rozumiem, że chcesz **wzmocnić ciąg adresowy**, rozpatrując go nie tylko w **obu kierunkach (od początku do końca i od końca do początku)**, ale także **od miejsca wystąpienia** każdej cyfry. To podejście wprowadza **dodatkową warstwę interpretacji**, gdzie każda cyfra w ciągu jest analizowana zarówno w kontekście **globalnym** (cały ciąg), jak i **lokalnym** (jej bezpośrednie sąsiedztwo). Postaram się teraz opisać tę metodę bardziej szczegółowo.

## 1. Interpretacja ciągu w obu kierunkach

Ciąg adresowy (np. 0123456789012345678901234 ) jest **hierarchicznie interpretowany** w dwóch głównych kierunkach:

1. **Od początku do końca (→)**: Każda cyfra definiuje **kolejny poziom szczegółowości** w opisie układu pikseli.
2. **Od końca do początku (←)**: Każda cyfra definiuje **kontekst globalny** (np. relacje między kolorami).

**Przykład ciągu:** 0123456789012345678901234

- **Od początku do końca:**
  - **0**: Biel (podzbiór główny).
  - **1**: Czarny (podzbiór w ramach bieli).
  - **2**: Czerwony (podzbiór w ramach czarnego).
  - ...
- **Od końca do początku:**
  - **4**: Relacja między kolorami na najniższym poziomie szczegółowości.
  - **3**: Relacja między kolorami na wyższym poziomie.
  - ...

## 2. Interpretacja od miejsca wystąpienia

Każda cyfra w ciągu jest **dodatkowo analizowana** w kontekście jej **miejsca wystąpienia**:

1. **Lokalny kontekst**: Każda cyfra jest interpretowana w odniesieniu do **sąsiednich cyfr** (np. poprzedniej i następnej).
2. **Globalny kontekst**: Każda cyfra jest interpretowana w odniesieniu do **całego ciągu**.

**Przykład:**

Dla ciągu 0123456789... , cyfra **5** na pozycji **6** jest analizowana:

- **Lokalnie**: W kontekście sąsiednich cyfr **4** i **6**.
- **Globalnie**: W kontekście całego ciągu (np. jako część podciągu 012345 ).

## 3. Wzmocnienie ciągu

Aby **wzmocnić ciąg**, wprowadzamy **dodatkowe funkcje interpretacyjne**, które uwzględniają:

1. **Lokalne relacje**: Analizują każdą cyfrę w kontekście jej **sąsiedztwa**.
2. **Globalne relacje**: Analizują każdą cyfrę w kontekście **całego ciągu**.

3. **Kierunkowość:** Interpretuj ciąg zarówno **od początku do końca**, jak i **od końca do początku**.

#### Przykład funkcji wzmacniających:

- **Funkcja lokalna:** Dla cyfry  $a_k$ , oblicz jej znaczenie na podstawie  $a_{k-1}$  i  $a_{k+1}$ .
- **Funkcja globalna:** Dla cyfry  $a_k$ , oblicz jej znaczenie na podstawie  $a_1 a_2 \dots a_{25}$ .

## 4. Kodowanie i dekodowanie

#### Kodowanie:

1. **Ekstrakcja cech:** Dla każdego poziomu  $k$ , oblicz wartość  $a_k = f_k(X)$ , gdzie  $X$  to obraz.
2. **Analiza lokalna:** Dla każdej cyfry  $a_k$ , oblicz jej znaczenie w kontekście sąsiedztwa.
3. **Analiza globalna:** Dla każdej cyfry  $a_k$ , oblicz jej znaczenie w kontekście całego ciągu.
4. **Konkatenacja:** Zbuduj ciąg adresowy  $a_1 a_2 \dots a_{25}$  wraz z dodatkowymi informacjami o lokalnych i globalnych relacjach.

#### Dekodowanie:

1. **Dekompozycja ciągu:** Rozłóż ciąg na cyfry  $a_1 a_2 \dots a_{25}$  oraz informacje o lokalnych i globalnych relacjach.
2. **Rekonstrukcja cech:** Dla każdej cyfry  $a_k$ , odwróć funkcję  $f_k$ , aby odtworzyć cechy obrazu.
3. **Synteza obrazu:** Użyj np. autoenkoderów do generacji obrazu na podstawie odtworzonych cech.

## 5. Przykład praktyczny

**Krok 1:** Użytkownik wybiera obraz  $X$  do zakodowania.

**Krok 2:** System oblicza ciąg adresowy:

- $a_1 = f_1(X) = 0$  (biel),
- $a_2 = f_2(X, a_1) = 1$  (czarny),
- $a_3 = f_3(X, a_1 a_2) = 2$  (czerwony),
- ...

**Krok 3:** System analizuje każdą cyfrę w kontekście lokalnym i globalnym:

- Dla  $a_3=2$ , analizuje sąsiedztwo ( $a_2=1$  i  $a_4=3$ ) oraz kontekst globalny (np. podciąg 012).

**Krok 4:** System dekoduje obraz:

- Dla każdej cyfry  $a_k$ , odwróć funkcję  $f_k$  i odtwórz cechy.
- Zsumuj wyniki dekodowania, aby odtworzyć pełny obraz.

## 6. Podsumowanie

Proponowana metoda opiera się na:

1. **Interpretacji ciągu w obu kierunkach** (od początku do końca i od końca do początku).
2. **Analizie lokalnej i globalnej** każdej cyfry w ciągu.
3. **Wzmocnieniu ciągu** poprzez dodatkowe funkcje interpretacyjne.

To podejście pozwala na **precyzyjne odwzorowanie relacji między kolorami** oraz **efektywne kodowanie/dekodowanie** układów pikseli.

Interpretacji ciągu w obu kierunkach byłaby skuteczna jedynie dla uzyskania lub zakodowania wstępnego bo np mając jedną połowę ekranu białą drugą czarną to taki ciąg miałby postać 00000000001111111111 i aby zakodować ten obraz ciąg ten byłby rozpatrywany dla 0 (biały) i 1 (czarny) i byłby rozpoznany jako syntetyczny właśnie z tego że w obu kierunkach mamy do czynienia z powtarzalnością



Dokładnie! Jeśli interpretujemy ciąg w **obu kierunkach** (od początku do końca i od końca do początku) oraz zauważamy **powtarzalność** (np. ciąg 00000000001111111111), to możemy **skutecznie zakodować** taki obraz jako **syntetyczny** (czyli złożony z powtarzających się wzorców). W tym przypadku ciąg 00000000001111111111 reprezentuje **połowę ekranu białego i połowę ekranu czarnego**, co jest **łatwe do skompresowania** dzięki swojej **regularności**.

Teraz rozwinę tę ideę, aby pokazać, jak można **efektywnie kodować** takie syntetyczne obrazy, wykorzystując **własności ciągów** i **interpretację w obu kierunkach**.

## 1. Analiza ciągu w obu kierunkach

Ciąg 00000000001111111111 można interpretować w **obu kierunkach**:

### 1. Od początku do końca (→):

- Widzimy sekwencję **10 zer**, a następnie **10 jedynek**.
- Można to zakodować jako:  **$10 \times 0$ ,  $10 \times 1$** .

### 2. Od końca do początku (←):

- Widzimy sekwencję **10 jedynek**, a następnie **10 zer**.
- Można to zakodować jako:  **$10 \times 1$ ,  $10 \times 0$** .

### Wniosek:

W obu kierunkach ciąg jest **symetryczny** i **powtarzalny**, co pozwala na **skuteczną kompresję**.

## 2. Kodowanie syntetyczne

Dla ciągu 00000000001111111111 możemy zastosować **kodowanie syntetyczne**, które wykorzystuje **powtarzalność**:

### 1. Identyfikacja wzorca:

- Wzorzec to **0** (biały) i **1** (czarny).

### 2. Zliczenie powtórzeń:

- **$10 \times 0$ ,  $10 \times 1$** .

### 3. Zapis skompresowany:

- Zamiast zapisywać cały ciąg, zapisujemy tylko **wzorzec i liczbę powtórzeń**:  **$10 \times 0$ ,  $10 \times 1$** .

### Zysk kompresji:

- Oryginalny ciąg ma **20 znaków**.
- Skompresowany zapis ma **8 znaków** ( **$10 \times 0$ ,  $10 \times 1$** ).

## 3. Interpretacja w obu kierunkach dla kompresji

Interpretacja ciągu w **obu kierunkach** pozwala na **identyfikację powtarzalności**:

### 1. Od początku do końca:



- Sekwencja **10 zer** → wzorzec 0 powtórzony 10 razy.

## 2. Od końca do początku:

- Sekwencja **10 jedynek** → wzorzec 1 powtórzony 10 razy.

### Wniosek:

W obu kierunkach widzimy **tę samą powtarzalność**, co potwierdza, że ciąg jest **syntetyczny** (regularny).

## 4. Zastosowanie dla bardziej złożonych obrazów

Jeśli obraz jest bardziej złożony, ale nadal zawiera **powtarzalne wzorce**, możemy zastosować **podobną metodę**:

### 1. Podział obrazu na regiony:

- Podziel obraz na mniejsze bloki (np. 8x8 pikseli).

### 2. Analiza każdego bloku:

- Dla każdego bloku sprawdź, czy zawiera **powtarzalne wzorce**.

### 3. Kodowanie bloków:

- Jeśli blok jest syntetyczny (np. jednolity kolor), zakoduj go jako **wzorzec i liczbę powtórzeń**.
- Jeśli blok jest złożony, zastosuj **kodowanie szczegółowe** (np. zapis wartości pikseli).

### Przykład:

Dla obrazu z **czterema regionami**:

- Region 1: 0000000000 (10×0),
- Region 2: 1111111111 (10×1),
- Region 3: 0000000000 (10×0),
- Region 4: 1111111111 (10×1).

### Kodowanie:

- 10×0, 10×1, 10×0, 10×1 .

## 5. Optymalizacja dla syntetycznych obrazów

Aby **zoptymalizować kodowanie** dla syntetycznych obrazów, można zastosować:

### 1. Wykrywanie powtarzalności:

- Algorytmy wykrywające **sekwencje powtarzających się wartości** (np. Run-Length Encoding).

### 2. Hierarchiczne kodowanie:

- Podziel obraz na **poziomy szczegółowości** i koduj każdy poziom osobno.

### 3. Kompresja stratna:

- Dla obrazów syntetycznych można zastosować **kompresję stratną**, która ignoruje niewielkie różnice w wartościach pikseli.

## 6. Podsumowanie

Interpretacja ciągu w **obu kierunkach** jest **skuteczna** dla **obrazów syntetycznych** (z powtarzalnymi wzorcami), ponieważ:

1. **Wykrywa powtarzalność**: W obu kierunkach widzimy te same sekwencje.

2. **Umożliwia kompresję**: Powtarzalne wzorce można zakodować w sposób zwężły (np. 10×0, 10×1 ).

3. **Skaluje się dla złożonych obrazów:** Można ją zastosować do **regionów obrazu**, a nie tylko do całego ciągu.

Dzięki temu podejściu **obraz syntetyczny** (np. połowa biała, połowa czarna) może być **zakodowany bardzo efektywnie**, co jest szczególnie przydatne w **przetwarzaniu dużych zbiorów danych**.

a gdy mamy np 000001111111111100000 to wtedy po srodku ciagu mamy w obu kierunkach po 5 jedynek a nastepnie nic do konca czyli biel , a gdy mamy 10101010101010101010 to wtedy juz muzimy aby bylo prosto i bez udziwnien dla kodu wbic w srodek i juz zbior po zbiorze tym tokiem badac wystapienia bo kazda liczba jest inna, i w kontekście tego ze potrzeba tu o wiele wiekszego zbioru przy skomplikowanych obrazach ciag 25 pozycyjny tak naprawde niesie ze soba 75 pozycji poniewaz w obu kierunkach to jest  $2 \times 25$  i potem dwie dywidendy od miejsca wystapienia to kolejne +25 czyli razem 75, potrzeba tylko sposobu aby do takiego wypadkowego wyniku 25 wedrowaly ciagi z przedzialow 75