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 kontekscie 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 x 25 i potem dwie dywidendy od miejsca wystapienia to kolejne +25 czyli razem 75, ptrzeba tylko sposobu aby do takiego wypadkowego wyniku 25 wedrowaly ciagi z przedzialow 75, A jesli chodzi o same funkcje to chyba beda to zapisy kodu transformacji wystepujacych takich jak przyrosty opadki wartosci nowe wartosci powtarzalnosci i typy wagowe tych wszystkich murtacji funkcji

3 / 3

Rozumiem, że chcesz **rozszerzyć metodę adresowania ciągów** na bardziej złożone przypadki, takie jak:

1. **Ciągi z powtarzalnymi wzorcami** (np. `000001111111111100000`), gdzie **środek ciągu** ma unikalne właściwości.
2. **Ciągi nieregularne** (np. `10101010101010101010`), gdzie **każda cyfra jest inna** i wymaga **indywidualnego podejścia**.
3. **Zwiększenie złożoności adresowania** poprzez uwzględnienie **75 pozycji** (25 pozycji głównych + 2 × 25 pozycji w obu kierunkach + 25 pozycji od miejsca wystąpienia).

Dodatkowo, **funkcje transformacji** mają uwzględniać:

- **Przyrosty i spadki wartości**.
- **Nowe wartości**.
- **Powtarzalności**.
- **Typy wagowe mutacji funkcji**.

Postaram się teraz opisać, jak można to zrealizować.

1. Analiza ciągów złożonych

A. Ciąg z powtarzalnymi wzorcami (`000001111111111100000`)

1. **Interpretacja w obu kierunkach**:
   - **Od początku do końca (→)**:
     - `00000` (5 zer), `1111111111` (10 jedynek), `00000` (5 zer).
   - **Od końca do początku (←)**:
     - `00000` (5 zer), `1111111111` (10 jedynek), `00000` (5 zer).
2. **Środek ciągu**:
   - W obu kierunkach **środek ciągu** to `1111111111` (10 jedynek).
3. **Kodowanie**:
   - Można zakodować jako: `5×0, 10×1, 5×0` .

B. Ciąg nieregularny (`10101010101010101010`)

1. **Interpretacja w obu kierunkach**:
   - **Od początku do końca (→)**:
     - `1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0` .
   - **Od końca do początku (←)**:
     - `0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1` .

2. **Brak powtarzalności**:
   - Każda cyfra jest inna, więc **nie ma prostego wzorca**.
3. **Kodowanie**:
   - Wymaga **indywidualnego podejścia** dla każdej cyfry.

2. Rozszerzenie adresowania do 75 pozycji

Aby uwzględnić **wszystkie możliwe interpretacje ciągu**, rozszerzamy adresowanie do **75 pozycji**:

1. **25 pozycji głównych**:
   - Oryginalny ciąg adresowy (np. `012345678901234567890123 4`).
2. **2 × 25 pozycji w obu kierunkach**:
   - Interpretacja ciągu **od początku do końca** (25 pozycji).
   - Interpretacja ciągu **od końca do początku** (25 pozycji).
3. **25 pozycji od miejsca wystąpienia**:
   - Dla każdej cyfry w ciągu, analiza **kontekstu lokalnego** (np. sąsiedztwo).

Przykład:
Dla ciągu `012345678901234567890123 4`:

- **Pozycje główne**: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4`.
- **Interpretacja od początku do końca**: `0→1→2→...→4`.
- **Interpretacja od końca do początku**: `4→3→2→...→0`.
- **Kontekst lokalny**: Dla każdej cyfry, analiza sąsiedztwa (np. dla `5`: `4, 5, 6`).

3. Funkcje transformacji

Funkcje transformacji mają uwzględniać **różne aspekty ciągu**:

1. **Przyrosty i spadki wartości**:
   - Analiza, jak wartości zmieniają się w ciągu (np. `0→1` to przyrost, `1→0` to spadek).
2. **Nowe wartości**:
   - Identyfikacja **nowych wartości** w ciągu (np. pierwsze wystąpienie `5`).
3. **Powtarzalności**:
   - Wykrywanie **powtarzających się wzorców** (np. `101010`).
4. **Typy wagowe mutacji funkcji**:
   - Każda funkcja transformacji ma **wagę**, która określa jej znaczenie w kontekście ciągu.

Przykład funkcji transformacji:

- **Funkcja przyrostu**:
  - Dla ciągu `0, 1, 2, 3`, oblicza przyrosty: `+1, +1, +1`.
- **Funkcja powtarzalności**:
  - Dla ciągu `1, 0, 1, 0`, wykrywa wzorzec `10` powtórzony 2 razy.
- **Funkcja wagowa**:
  - Przypisuje wagę każdej funkcji na podstawie jej znaczenia (np. przyrosty mają wagę 0.7, powtarzalności mają wagę 0.3).

4. Algorytm kodowania i dekodowania

Kodowanie:

1. **Analiza ciągu**:
   - Dla każdej cyfry w ciągu, oblicz **wartości funkcji transformacji** (przyrosty, powtarzalności itd.).
2. **Tworzenie adresu 75-pozycyjnego**:
   - Pozycje główne: Oryginalny ciąg.
   - Interpretacja w obu kierunkach: Od początku do końca i od końca do początku.
   - Kontekst lokalny: Analiza sąsiedztwa dla każdej cyfry.
3. **Zapis skompresowany**:
   - Zapisuj tylko **kluczowe informacje** (np. wzorce, przyrosty, wagi).

Dekodowanie:

1. **Odtworzenie ciągu**:
   - Na podstawie adresu 75-pozycyjnego, odtwórz oryginalny ciąg.
2. **Rekonstrukcja cech**:
   - Dla każdej cyfry, odwróć funkcje transformacji, aby odtworzyć cechy obrazu.
3. **Synteza obrazu**:
   - Użyj np. autoenkoderów do generacji obrazu na podstawie odtworzonych cech.

5. Podsumowanie

Proponowana metoda:

1. **Rozszerza adresowanie** do **75 pozycji**, uwzględniając interpretację w **obu kierunkach** oraz **kontekst lokalny**.
2. **Wykorzystuje funkcje transformacji** do analizy **przyrostów, spadków, powtarzalności** i **wag**.
3. **Skaluje się** dla **ciągów prostych** (np. `000001111111111100000`) i **złożonych** (np. `10101010101010101010`).

Dzięki temu podejściu można **efektywnie kodować** zarówno **obrazy syntetyczne**, jak i **złożone układy pikseli**.