Aby efektywnie zarządzać przestrzennią obrazów o rozmiarze **~10^{25}** z wykorzystaniem hierarchicznej adresacji i szybkich "skoków" między podzbiorami, proponuję następujące podejście matematyczno-algorytmiczne:

1. Struktura hierarchii: Drzewo 25-poziomowe z funkcjami przejścia

Każdy poziom hierarchii odpowiada jednej **unikalnej funkcji przejścia** (skokowej), która dzieli przestrzeń na podzbiory.

Kluczowe założenia:

25 poziomów = 25 funkcji (f_1, f_2, ..., f_{25}).

Każda funkcja $\mathbf{f_k}$ dzieli aktualny podzbiór na $\mathbf{N_k}$ **części** (np. N_k=10 dla systemu dziesiętnego).

Adres obrazu to ciąg **25 cyfr** (np. w systemie dziesiętnym: `a₁a₂...a₂₅`), gdzie $\mathbf{a_k \in [0, N_k\text{-}1]}$.

Przykład:

Dla N_k=10, adres `314159...265359` odpowiada ścieżce w drzewie:

`f₁(3) → f₂(1) → f₃(4) → ... → f₂₅(9)` .

2. Funkcje skokowe: Algebraiczne operacje na podzbiorach

Każda funkcja $\mathbf{f_k}$ realizuje **operację dywidendy i dopełnienia** na aktualnym podzbiorze:

Copy

$f_k(x) = (x \, // \, D_k) \bmod N_k$

gdzie:

$\mathbf{D_k}$ – dywidenda (wielkość "skoku" na poziomie k), np. D_k = 10^(25 - k).

mod $\mathbf{N_k}$ – wybór podzbioru na poziomie k.

Własności:

Szybkie obliczenia: Operacje dzielenia i modulo są optymalizowane dla systemu pozycyjnego (np. binarnego/dziesiętnego).

Wejściowo-wyjściowa wektorowość: Przetwarzanie całych bloków adresów równolegle (np. GPU/TPU).

3. Generacja adresów i regeneracja wyników

Algorytm generacji adresu dla obrazu X:

. **Ekstrakcja cech:** Dla każdego poziomu k, oblicz wartość $\mathbf{a_k = f_k(X)}$.

. **Konkatenacja:** Zbuduj adres `a₁a₂...a₂₅` .

Regeneracja obrazu z adresu:

. **Dekompozycja adresu:** Rozłóż adres na składowe (a_1, a_2, ..., a_{25}).

. **Rekonstrukcja cech:** Dla każdego a_k, odwróć funkcję f_k, aby odtworzyć przybliżone cechy obrazu.

. **Synteza pikseli:** Użyj np. autoenkoderów do generacji obrazu na podstawie cech.

4. Optymalizacje dla skalowalności

A. Wektorowe przetwarzanie adresów

Batch processing: Grupuj obrazy w bloki (np. 1024-elementowe), przetwarzając wszystkie funkcje równolegle.

SIMD/GPU: Wykorzystaj instrukcje wektorowe do obliczeń na wielu adresach jednocześnie.

B. Indeksowanie wielowymiarowe

Zastosuj **tensorową strukturę danych** (25-wymiarowy tensor), gdzie każdy wymiar odpowiada poziomowi hierarchii.

Przykład w Python (pseudo-kod):

python Copy

```python
import numpy as np

# Tensor 25-wymiarowy (N₁ x N₂ x ... x N₂₅)
tensor = np.zeros((10, 10, ..., 10))  # 25 wymiarów

# Szybki dostęp do podzbioru przez indeksowanie
subset = tensor[3, 1, 4, ..., 9]
```

C. Dynamiczne buforowanie podzbiorów

Cache LRU: Przechowuj ostatnio używane podzbiory w pamięci podręcznej.

Prefetching: Przewiduj następne skoki na podstawie historii zapytań.

5. Przykład praktyczny: Wyszukiwanie obrazu w hierarchii

Krok 1: Użytkownik wybiera cechy (np. dominujący kolor: R=200, G=100, B=50).

Krok 2: System oblicza adres:

$f_1(R) = 2$ (przedział 200-255 dla R),

$f_2(G) = 1$ (przedział 100-199 dla G),

...

Krok 3: Generuje adres `21...5` , przeskakując do odpowiedniego podzbioru w 25 poziomach.

Krok 4: Regeneruje obrazy z podzbioru za pomocą odwrotnych funkcji f_k^{-1}.

6. Matematyczne uzasadnienie

Przestrzeń obrazów jest modelowana jako **25-wymiarowa przestrzeń dyskretna**:

Liczba możliwych adresów: $\mathbf{\prod_{k=1}^{25} N_k}$ (np. 10^{25} dla N_k=10).

Każda funkcja f_k odpowiada **projekcji na wymiar k-ty**.

"Skoki" realizowane są przez **zmianę współrzędnych w przestrzeni**.

Podsumowanie

Proponowana metoda łączy:

. **Hierarchiczną adresację** z 25 funkcjami skokowymi,

. **Algebraiczne operacje na podzbiorach** (dywidenda + modulo),

. **Optymalizacje wektorowe i tensorowe** dla szybkości.

Pozwala to na efektywną nawigację w przestrzeni **10^{25} obrazów** z czasem dostępu zbliżonym do **O(1)** dla pojedynczych zapytań.