Elaborazione delle immagini

By Achraf Eddari 45766A
https://methods-image-processing.vercel.app/upload

Presentazione per l’esame di METHODS FOR IMAGE PROCESSING

Introduzione & Obiettivi

Introduzione all'elaborazione delle immagini

- **Definizione**
- **Obiettivi principali**
- **La mia implementazione**

Negativo di un'Immagine

$$s=(L-1)-r$$

Dove r è l'intensità corrente del pixel e L è il massimo valore di intensità (per immagini a 8 bit, L=255).

Trasformazioni Gamma

$$s = c \cdot (r^{\gamma})$$

Dove c è una costante e γ è il parametro che determina la forma della curva.

Equalizzazione dell’istogramma

Processo

- Calcolare la **funzione di distribuzione cumulativa** (CDF) dell'istogramma.
- Usare la CDF per mappare le intensità originali a nuove intensità, migliorando la percezione visiva

Codifica di Huffman

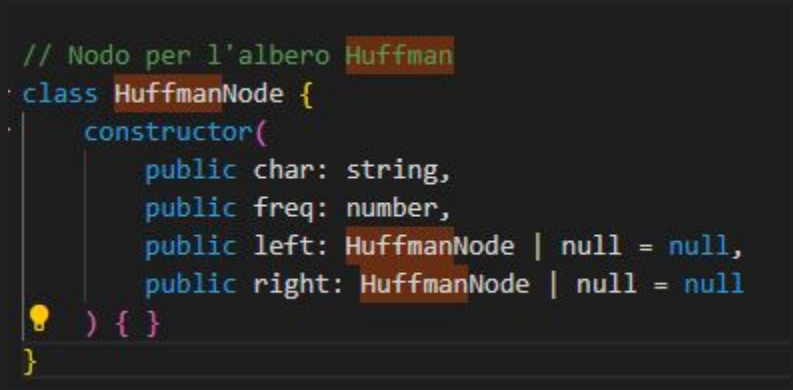
Processo

- **Fase 1**: Calcolare la frequenza di ogni simbolo nell'immagine.
- **Fase 2**: Costruire l'albero di Huffman.
- **Fase 3**: Assegnare codici binari a ogni simbolo.

HuffmanNode: Creazione del Nodo dell'Albero

La classe `HuffmanNode` rappresenta un nodo dell'albero di Huffman. Ogni nodo ha:

- Un carattere o un valore di intensità pixel (`char`).
- La sua frequenza di occorrenza (`freq`).
- Punti di riferimento a nodi figli a sinistra (`left`) e a destra (`right`), che servono a costruire l'albero binario.



buildHuffmanTree: Costruzione dell'Albero di Huffman

Questa funzione costruisce l'albero di Huffman

- **Passaggio 1**: Creazione di un array di nodi usando le frequenze di ciascun pixel dell'immagine.
- **Passaggio 2**: L'array viene ordinato in base alla frequenza.
- **Passaggio 3**: Si combinano i nodi con frequenze più basse per formare un nuovo nodo con la somma delle due frequenze, finché rimane un solo nodo radice.

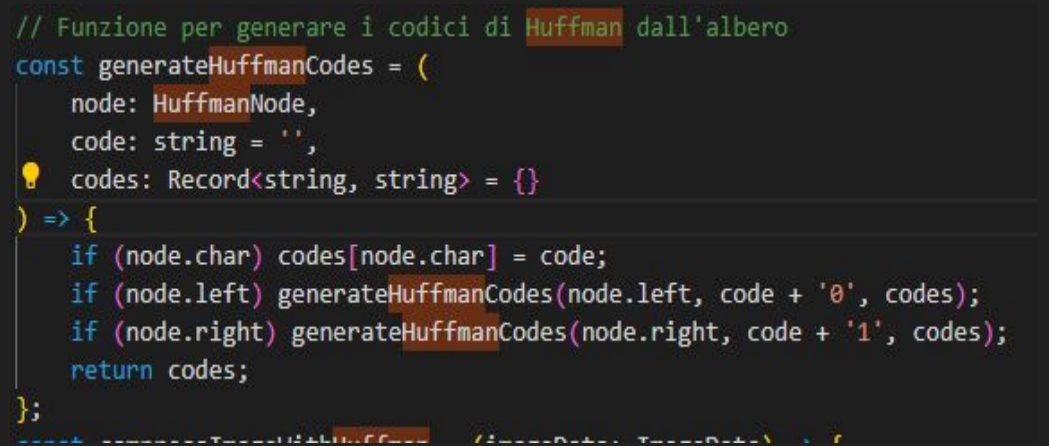
```
// Funzione per costruire l'albero di Huffman
const buildHuffmanTree = (freqMap: Record<string, number>) => {
    const nodes = Object.entries(freqMap).
    map(([char, freq]) => new HuffmanNode(char, freq));

    while (nodes.length > 1) {
        nodes.sort((a, b) => a.freq - b.freq);
        const left = nodes.shift();
        const right = nodes.shift();
        if (left && right) {
            const newNode =
            new HuffmanNode('', left.freq + right.freq, left, right);
            nodes.push(newNode);
        }
    }
    return nodes[0];
};
```

generateHuffmanCodes: Generazione dei Codici di Huffman

Una volta costruito l'albero di Huffman, questa funzione ricorsiva viaggia lungo l'albero per generare i codici binari per ogni valore di pixel. Durante la navigazione:

- Spostandosi a sinistra si aggiunge uno `0` al codice binario.
- Spostandosi a destra si aggiunge uno 1 al codice.
- Alla fine, viene creato un dizionario che mappa ogni combinazione di valori pixel al suo codice Huffman.



```
// Funzione per generare i codici di Huffman dall'albero
const generateHuffmanCodes = (
    node: HuffmanNode,
    code: string = '',
    codes: Record<string, string> = {}
) => {
    if (node.char) codes[node.char] = code;
    if (node.left) generateHuffmanCodes(node.left, code + '0', codes);
    if (node.right) generateHuffmanCodes(node.right, code + '1', codes);
    return codes;
};
```

compressImageWithHuffman: Compressione

Questa funzione comprime l'immagine usando l'algoritmo di Huffman:

- **Passaggio 1**: Viene calcolata la frequenza di ciascun valore pixel (basato sui valori RGB dei pixel).
- **Passaggio 2**: Si costruisce l'albero di Huffman e si generano i codici di compressione.
- **Passaggio 3**: Si percorre l'immagine originale e, per ogni pixel, si sostituisce il valore RGB con il suo codice di Huffman corrispondente.
- **Passaggio 4**: Si calcolano i byte compressi e si confrontano con i byte originali per determinare l'efficacia della compressione.

```
// Crea l'albero di Huffman e genera i codici
const huffmanTree = buildHuffmanTree(freqMap);
const huffmanCodes = generateHuffmanCodes(huffmanTree);

// Comprimi l'immagine usando i codici di Huffman
let compressedData = '';
for (let i = 0; i < imageData.data.length; i += 4) {
    const r = imageData.data[i];
    const g = imageData.data[i + 1];
    const b = imageData.data[i + 2];
    const pixelValue = `${r},${g},${b}`;
    compressedData += huffmanCodes[pixelValue];
}

// Calcolo del numero di byte nel dato compresso
const originalBytes = imageData.data.length; // Dimensione originale in byte
const compressedBytes = Math.ceil(compressedData.length / 8); // Dimensione compressa in byte

console.log(`Original Bytes: ${originalBytes}, Compressed Bytes: ${compressedBytes}`);
```

Conclusion

- AI
- Riconoscimento di shoes