La Recherche d'Images Similaires
Projet intégrateur

Partie #4: Fonctionnelle (Scheme)

uOttawa

CSI2520[A] Paradigmes de programmation

Présenté pour Robert Laganière

Yassine Moumine 300140139

Date: le 10 Avril 2024

Table de Matiere

Table de Figures

Introduction

Dans le monde moderne, la prolifération rapide des images nécessite des outils informatiques avancés pour analyser, rechercher, classer et découvrir des images d'intérêt. Ce projet vise à implémenter une méthode de recherche d'images similaires en utilisant la **programmation fonctionnelle Scheme**.

Données d'Entrée

La base de données d'images est fournie au format JPG, avec les histogrammes précalculés enregistrés dans des fichiers texte. En outre, 16 images de requête sont fournies au format JPG et PPM. Pour cette partie nous utilisons **le format jpg texte** qui contient les histogrammes des images précalculées.

Objectifs:

- Résoudre le problème de similarité en décodant les histogrammes des images.
- Utiliser la programmation fonctionnelle pour trouver les 5 images les plus similaires à chaque image de requête

Solution Algorithmique

1. (countPixels '(histogram values)): Calcule le nombre total de pixels représentés dans un histogramme.

```
(define (countPixels histogram)
  (let loop-count ((remainingHistogram histogram)
(total 0))
    (if (null? remainingHistogram)
        total
        (loop-count (cdr remainingHistogram) (+ (car
remainingHistogram) total)))))
```

Figure 1: Code de la fonction (countPixels '(histogram values))

2. (normalize '(histogram values) totalPixels): Normalise un histogramme en divisant chaque valeur par le nombre total de pixels.

```
(define (normalize histogram totalPixels)
  (map (lambda (value) (/ value totalPixels))
histogram))
```

Figure 2: Code de la fonction (normalize '(histogram values) totalPixels)

3. (hist-intersect '(query histogram) '(dataset histogram)): Calcule l'intersection entre deux histogrammes.

```
(define (hist-intersect queryHistogram
datasetHistogram)
  (let loop-intersect ((qHist queryHistogram) (dHist
datasetHistogram) (intersectionSum 0))
    (if (null? qHist)
        intersectionSum
        (loop-intersect (cdr qHist) (cdr dHist) (+
intersectionSum (min (car qHist) (car dHist)))))))
```

Figure 3: Code de la fonction (hist-intersect '(query histogram) '(dataset histogram))

4. (sort-by-value '((filename1 val1) (filename2 val2) ...)): Trie une liste de paires par valeur dans un ordre décroissant.

```
(define (sort-by-value pairsList)
  (sort pairsList (lambda (pair1 pair2) (> (cdr pair1)
(cdr pair2)))))
```

Figure 4: Code de la fonction (sort-by-value '((Pair1) (Pair2)))

5. (displayTopFive similarityScores): Affiche les cinq premières images basées sur leur score de similarité.

```
(define (displayTopFive similarityScores)
  (for-each (lambda (score) (displayln score)) (take
similarityScores 5)))
```

Figure 5: Code de la fonction (displayTopFive similarityScores)

6. (similaritySearch "queryImages/file.jpg.txt" "imageDataset2_15_20/"): Effectue une recherche de similarité entre un histogramme de requête et un ensemble de données d'histogrammes, affichant les 5 images les plus similaires.

```
(define (similaritySearch queryHistogramFile
datasetDirectory)
  (let ((queryHist (read-hist-file queryHistogramFile))
        (datasetFiles (list-text-files-in-directory
datasetDirectory)))
    (let ((scores (map (lambda (datasetFile)
                         (let ((datasetHist
(read-hist-file (string-append datasetDirectory
datasetFile))))
                           (cons datasetFile
(hist-intersect (normalize queryHist (countPixels
queryHist))

(normalize datasetHist (countPixels datasetHist))))))
                       datasetFiles)))
      (displayTopFive (sort-by-value scores)))))
```

Figure 6: Code de la fonction (similaritySearch "queryImages/file.jpg.txt" "imageDataset2_15_20/")

Résultat Attendu

1. L'objectif est de trouver les 5 images les plus similaires à chaque image de requête, en utilisant l'intersection de leurs histogrammes comme vu précédemment dans la partie object-oriented, concurrence et logique.

```
> (similaritySearch "queryImages/q01.jpg.txt" "imageDataset2_15_20/")
(3588.jpg.txt . 10951/19200)
(2536.jpg.txt . 19397/34560)
(3553.jpg.txt . 1717/3200)
(1875.jpg.txt . 3043/5760)
(2592.jpg.txt . 87259/172800)
> (similaritySearch "queryImages/q13.jpg.txt" "imageDataset2_15_20/")
(3225.jpg.txt . 1)
(2083.jpg.txt . 42757/86400)
(2082.jpg.txt . 5341/10800)
(2078.jpg.txt . 5281/10800)
(2607.jpg.txt . 27209/57600)
> (similaritySearch "queryImages/q04.jpg.txt" "imageDataset2_15_20/")
(1799.jpg.txt . 36569/43200)
(848.jpg.txt . 85259/172800)
(4261.jpg.txt . 84511/172800)
(1800.jpg.txt . 10183/21600)
(1937.jpg.txt . 16177/34560)
```

Figure 7: Résultats de sortie

Réferences

[1] Teach Yourself Racket. (n.d.). Cs.uwaterloo.ca. https://cs.uwaterloo.ca/~plragde/flaneries/TYR/

[2] Welcome to Racket. (n.d.). Docs.racket-Lang.org. https://docs.racket-lang.org/guide/intro.html