BERNARD Julien
NOEL Alexandre

TP SAE - Manipulation d'images
Rapport

1) Solution personnelle

a) Idée de départ et le principe général

Cheminement de pensée:

- Version de base

Après avoir analyser le sujet, nous avons définis des grandes étapes de fonctionnement de l'application:

- Chargement de l'image,
- Calculs sur l'image,
    - Calcul de l'histogramme,
    - Sélection des couleurs représentatives,
    - Remplacement des couleurs par la couleurs la plus proche,
- Sauvegarde de l'image.

Nous avons donc fait une version 0 chargeant l'image, la parcourant en stockant dans une liste chaque couleur présent sur l'image associée avec le nombre d'occurrence (Map<Integer, Integer> histogramme). Pour chaque pixel de l'image, nous incrémentons la couleur correspondante (histogramme.put(nCoul, histogramme.getOrDefault(nCoul, 0) + 1)). Une fois l'histogramme créer, nous les trions par ordre d'occurrence décroissante puis nous en sélectionnons les premières (nombre de couleurs désiré). La liste de couleur ainsi obtenus nous permettait de remplacer chacuns des pixel de l'image d'origine par la couleur la plus proche.

- Axes d'améliorations

A partir de cet algorithme, nous avons élaboré deux axes d'améliorations:

- La création de l'histogramme,
- La sélection des couleurs représentatives.

A partir de ces axes d'amélioration, nous avons mis au point plusieurs programmes.

- Version 1

Dans la première version, nous nous sommes concentré sur le premier axe.
Pour se faire, nous avons décidé de regrouper les couleurs proches. Pour regrouper les couleurs, nous divisons la valeur de la couleur par un nombre définis à l'avance, puis nous arrondissons sa valeur à l'entier le plus proche, que nous multiplions par le même nombre que précédemment.

Ex: 120/36 = 3,3333 → 3,3333 = 3 → 3*36 = 108
100/36 = 2,7778 → 2,7778 = 3 → 3*36 = 108

Pour chaque groupe de couleurs ainsi construit, nous stockons le nombre d'occurence et nous procédons de la même façons que l'algorithme de base.

- Version 2:

Dans cette version, nous nous concentrons sur le deuxième axe d'amélioration. Pour améliorer cette solution, nous nous sommes aidé de formules mathématiques de statistique. (La création de l'histogramme est la même que l'algorithme de base).
Pour se faire, nous trions la liste de couleurs par les couleurs. Nous avons décomposé la liste ainsi triée en différents quantiles (nombre de couleurs +1).
A partir de ce point, nous pouvons définir les couleurs par les valeurs séparant chacun de ces quantiles. Cette méthode prend donc en compte les fréquences.
Une fois les couleurs obtenues, nous construisons donc la nouvelle image.

Le problème de cette version était que si la majorité des couleurs étaient des variantes d'une même couleur, nous retrouvions quasiment que cette couleur (et ses variantes) sur l'image.

- Version 3:

Pour pallier le problème constaté précédemment, nous avons décidé de ne plus pondérer la sélection des couleurs par la fréquence. Nous réalisons désormais un découpage de la liste sans prendre en compte la fréquence. La liste est donc découpée selon sa taille/ son nombre d'éléments (et non plus la somme de ses valeurs).

- Version 4:

Pour réaliser la version 4, nous avons procédé à une fusion de nos deux meilleures versions: la première et la troisième. Nous avons donc regroupé les couleurs (dans des intervals plus petits) avant de sélectionner des couleurs à interval régulier.

- Version 5:

Fusion entre la version 1 et la version 2 (= version 4 en prenant en compte les fréquences)

b) Algorithme

Algorithme générale des solutions

```
début fonction reductionCouleurs(image, nbCouleurs)
    histogramme ← rassemblerCouleurs(image)
    couleursTriées ← triHistogramme(histogramme)
    trierListeParFrequence(couleursTriées)
    couleursRepresentatives ← selectionCouleursRepresentatives(couleursTriées)
    imageReduite ← remplacerCouleurs(image, couleursRepresentatives)
    retourner imageReduite
fin
```

Algorithme de la fonction selectionCouleursRepresentatives pour la **solution 1**

```
début fonction selectionCouleursRepresentatives(histogramme)
    liste ← convertirMapEnListe(histogramme)
    listeTriées ← trierListeParFrequence(liste)
    couleursRepresentatives ← créerTableauDeCouleurs(nbCouleurs)
    pour i allant de 0 à nbCouleurs
        couleursRepresentatives[i] ← créerCouleur(listeTriées[i].getKey())
    fin pour
    retourner couleursRepresentatives
fin
```

Algorithme de la fonction selectionCouleursRepresentatives pour la **solution 3**

```
début fonction selectionCouleursRepresentatives(histogramme)
    couleursTriées ← trierCouleursParRGB(histogramme)
    tailleEchantillon ← taille(couleursTriées)
    ecartDivisions ← tailleEchantillon / nbCouleurs
    divisions ← créerTableauVide(nbCouleurs)
    sommeDivisionCourante ← 0
    indexDivisionCourante ← 0
    pour chaque entry dans couleursTriées:
        value ← couleursTriées.getKey()
        sommeDivisionCourante++
        si sommeDivisionCourante = ecartDivisions:
            divisions[indexDivisionCourante] ← value
            indexDivisionCourante++
            sommeDivisionCourante ← 0
       fin si
        si indexDivisionCourante >= nbCouleurs:
            sortir de la boucle
       fin si
    fin pour
    couleursRepresentatives ← créerTableauDeCouleurs(nbCouleurs)
    pour i allant de 0 à nbCouleurs:
        couleursRepresentatives[i] ← créerCouleur(divisions[i])
    fin pour
    retourner couleursRepresentatives
fin
```

c) Conception

Diagramme de classe :

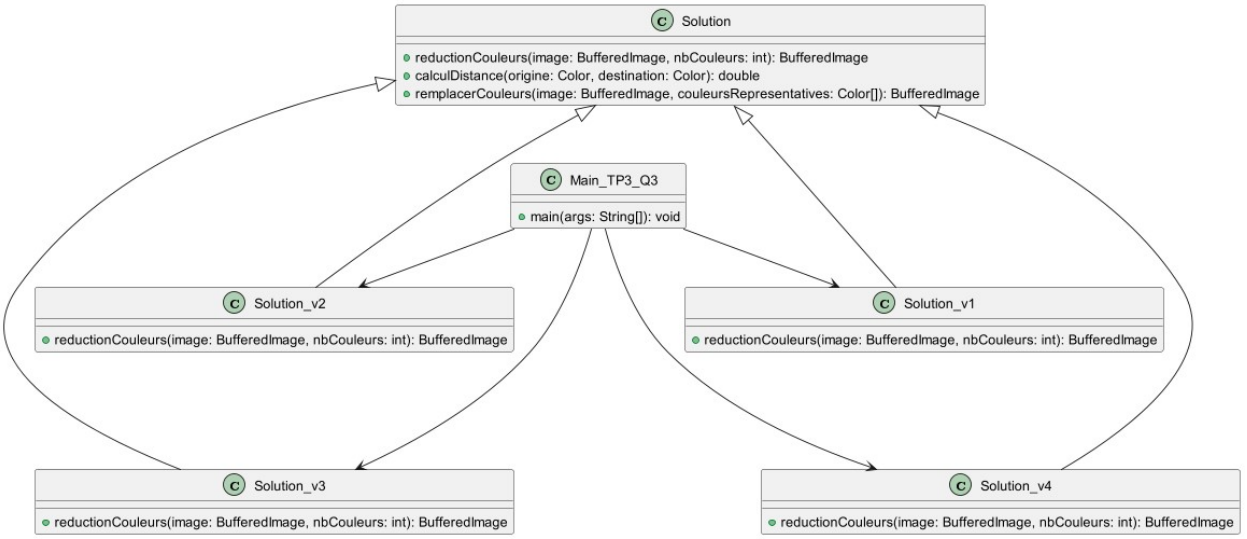
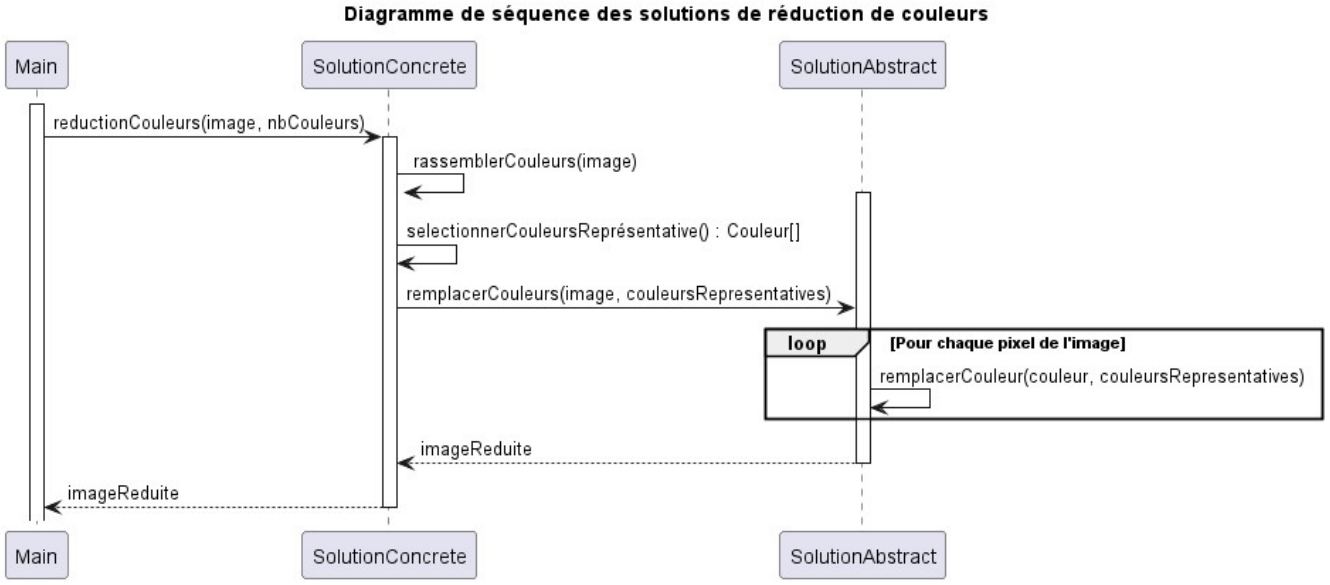


Diagramme de séquence général des solutions :



d) Utilisation de l'application

Pour lancer les 5 versions, il faut exécuter le main de la classe Main_TP3_Q3.java (dans le package solution_personnelle). Vous pouvez ajouter 2 paramètres pour spécifier le chemin de l'image et le nombre de couleurs restantes après la modification de l'image.

e) Amélioration de la solution

Des améliorations ont déjà été faites au cours du travail:

- Pour les versions regroupant les couleurs au début, nous regroupons à l'entier le plus proche au lieu de ne conserver que la partie entière.
- La version 3 était une amélioration de la version 2.
- Les versions 4 et 5 avaient pour but d'améliorer les versions précédentes.

Il y a aussi possibilité d'améliorer le regroupement des couleurs en changeant la façon dont c'est fait.
La suite du projet (partie SAE) nous a servi à améliorer la version 1.

2) Solution proposée dans la SAE

a) Principe de la solution

Le programme de base reste celui élaboré précédemment mais nous devons, désormais, combiner cet algorithme avec l'algorithme de KMeans. Pour se faire, nous réalisons une première sélection des couleurs, puis nous améliorons cette dernière en procédant à une boucle basée sur KMeans avant de l'afficher.

Pour la première sélection de couleurs, nous avions le choix:

- sélectionner les couleurs aléatoirement sur l'image,
- réalisé une première sélection des couleurs.

Nous avons choisi la première option. Pour ce faire, nous nous sommes servis du début de la version 1 élaboré précédemment: nous faisons un regroupement des couleurs et nous en sélectionnons les premières. Procédé permet ainsi de réduire le nombre de boucles de la partie suivante en réduisant le temps. Pour le vérifier nous avons comparé avec d'autres étudiants procédant de façon aléatoire:
En comparant les deux versions selon la durée et le nombre de boucles avant que l'image ne cesse de changer. Pour notre version, nous faisons 30 bouclés contre 100 pour l'autre groupe (chaque boucles étaient environ de la même durée pour les deux groupes). La durée de l'initialisation est faible. En résumé, notre version était 2 à 2,5 fois plus rapide que l'autre algorithme.

Après la première sélection des couleurs, nous réalisons plusieurs boucles servant à améliorer la sélection des couleurs. Pour se faire, nous associons a chaque couleurs sélectionnées une liste de couleur issue de l'image. Ces listes correspondent à toutes les couleurs de l'image dont la distance est la plus faible. (pour chaque pixel, on compare la distance avec chaques couleurs de la liste de référence et on l'associe avec celle qui a la distance la plus faible). Une fois les regroupements faits, on redéfinit chaque couleurs de référence par la moyenne de toutes les couleurs auxquelles elle est associée. Puis nous recommençons la boucle un certain nombre de fois avant d'afficher l'image.

b) Algorithme

Algorithme de réduction d'image

```
début fonction reductionCouleurs (image, nbCouleurs, nbRepetitions)
   tabCouleurs ← choixCouleursInitiales() (même méthode de selection de couleurs que la solution 1)
   pour i allant de 0 à nbRepetitions faire
      tabCouleurs ← KMeans(image, tabCouleurs, nbCouleurs)
   fin pour
   imageReduite ← remplacerCouleurs(image, tabCouleurs)
   retourner imageReduite
fin
```

Algorithme KMeans

```
début fonction KMeans (image, couleursRepresentatives, nbCouleurs)
        pixelColor ← obtenirCouleur(image, x, y)
            closestColor ← couleursRepresentatives[0]
            minDistance ← calculerDistance(pixelColor, closestColor)

            pour i allant de 1 à taille(couleursRepresentatives):
                distance ← calculerDistance(pixelColor, couleursRepresentatives[i])
                si distance < minDistance:
                    minDistance ← distance
                    closestColor ← couleursRepresentatives[i]

            temp ← obtenirListeCouleur(histogramme, closestColor)
            temp.ajouter(obtenirCouleur(image, x, y))
            mettreAJourListeCouleur(histogramme, closestColor, temp)

    couleursRepresentatives2 ← créerTableauDeCouleurs(nbCouleurs)
    index ← 0

    pour couleurRepres dans clésDe(histogramme):
        listeCouleurCorrespondantes ← obtenirValeur(histogramme, couleurRepres)
        sumRed, sumGreen, sumBlue, count ← 0

        pour couleur dans listeCouleurCorrespondantes:
            couleurCorrespondante ← créerCouleur(couleur)
            sumRed += couleurCorrespondante.getRed()
            sumGreen += couleurCorrespondante.getGreen()
            sumBlue += couleurCorrespondante.getBlue()
            count++

        moyenneRed ← sumRed / count
        moyenneGreen ← sumGreen / count
        moyenneBlue ← sumBlue / count

        couleursRepresentatives2[index] ← créerCouleur(moyenneRed, moyenneGreen, moyenneBlue)
        index++

    retourner couleursRepresentatives2
fin
```

c) Utilisation de l'application

Pour exécuter l'application il faut exécuter le main de la classe MainCluster.java (dans le package solution_SAE). Vous pouvez paramétrer le nombre de couleurs, le chemin de l'image et l'extension de l'image, de la ligne 14 à 17, avec les variables correspondes (nbCouleurs, img, extension).

d) Amélioration de la solution

Possibilité d'améliorer le nombre d'occurrence de l'algorithme de K-Means (déterminer à partir de quel moment il est inutile de continuer automatiquement). Corrigé ce problème permettrait d'éviter que la dernière itération soit moins bonne que l'avant dernière (car il est possible que l'image se dégrade légèrement à un moment).

Possibilité d'améliorer la sélection d'origine des couleurs. Nous n'avons pas réfléchi à de meilleures solutions, mais il s'agit probablement d'un des axes de réflexion possibles. En sachant que notre solution actuelle reste très bonne.

3) Tests et résultats

a) Base de tests utilisées

Nous avons utilisé les images suivantes :

- image de fleur (copie.png) car il y avait peu de couleurs différentes et des résultats étaient fournis.
- image d'ours (animaux/ours.png) car il y avait beaucoup de détail
- image de dauphin (animaux/dauphin_smll.png) car il y avait un dégradé de bleu
- image du logo strangerthings car peu de couleurs différentes
- image de peinture de Van Gogh car il y a beaucoup de nuances et de couleurs

b) Résultats obtenues

Image / nbcouleurs		V1	V2	V3	V4	V5	V K-Means
Fleure / 10	durée (ms)	231	228	205	274	153	2376
	Classement*	3	2	5	6	4	1
Fleure / 20	durée	275	290	233	304	291	2682
	Classement	6	4	2	5	3	1
Ours / 10	durée	202	466	312	414	143	2034
	Classement	3	2	4	6	5	1
Ours / 20	durée	244	422	448	358	187	2377
	Classement	6	5	3	2	4	1
Van Gogh / 10	durée	226	424	414	370	147	2199
	Classement	6	2	3	5	4	1
Van Gogh / 20	durée	244	468	387	419	181	2968
	Classement	6	3	2	4	5	1

*classement: établi selon la ressemblance, calculés selon la distance la plus basse entre l'image d'origine et obtenue

c) Analyse et conclusion

1) Attentes en terme de résultat (par rapport à ce que vous avez compris des algorithmes)

En théorie, dans tous les cas, l'algorithme de K-Means est censé être meilleurs que les précédents algorithmes. En effet, ce dernier réalise la boucle jusqu'à obtenir les couleurs les plus optimales. En revanche, ce dernier devrait mettre plus de temps à s'exécuter car il y a un plus grand nombre d'opérations.
En revanche, il est compliqué de savoir en avance lequel des autres algorithmes sera le meilleur.

2) Observations

On observe que la méthode K-Means est la plus performante parmi les méthodes évaluées, fournissant des indices de ressemblance généralement bas et montrant une meilleure cohérence dans la capture de la similitude avec les images d'origine. Néanmoins, elle prend également plus de temps d'exécution que les autres méthodes.

3) Conformité des résultats avec nos attentes

L'un des résultats le plus surprenant concerne la première version. En effet, dans plusieurs cas, elle possède une meilleure ressemblance visuellement, mais le calcul de la différence est l'un des plus grands.
Comme prévu, l'algorithme de K-Means offre les meilleurs résultats mais est le plus coûteux et le plus long. Certains des autres algorithmes offrent une très bonne ressemblance avec peu de temps de calculs.

4) Emettre les hypothèses qui pourraient expliquer des différences entres les résultats attendus et obtenus ;

Pour le cas de la première version, la différence peut s'expliqué car le choix des couleurs peut etre innexacte pour beaucoup de couleur, mais faiblement, ou alors car certaines couleurs ne sont pas présente et il y a un gros écarts de couleurs entre le pixel d'origine et celui d'arrivée (ex: absence d'orange dans la fleure).

d) Manière de lancer les tests

Vous devez exécuter le main de la classe LancementTests (dans le package tests). Pour changer l'image de test et le nombre de couleur souhaitées, vous pouvez les spécifier en arguments du main.