Compte rendu Projet 2: Mona Lisa

Fait par Remy XU, numéro etudiant: 22123345.

Preambule:

Le projet a été réalisé en Java et comporte 6 fichiers en .java. Les explications ont été faites principalement en javadoc et en commentaires. La méthode main se situe dans la classe Exemple.

Elle prend en compte les images de couleurs.

Liste des choses qui ont été faites:

- Ouverture d'une image et sa transformation en tableau de pixels RGB
- Conversion de l'image RGB en CIE lab
- Création de la structure de graphe, arêtes, sommets, voisin de sommets etc…
- Création des arêtes en 8-connexité ou moins selon la situation du sommet.
- Création de poids entre les valeurs Lab des pixels (sommets du graphe).
- Implémentation de l'algorithme de Dijkstra entre deux coordonnées. (Fonctionne correctement pour une distance des x et y d'environ 100 pixels. Au-delà, pas de garantie de fonctionner ou le programme tourne pendant trop longtemps.)
- Des exemples qui permettent de montrer le bon fonctionnement de ce qui a été vu plus haut.

Conversion RGB en CIE L*a*b*:

On rappelle ici que pour mon programme on prendra des valeurs entre 0 et 255. 0,0,0 pour noir et 255,255,255 pour blanc malgre la norme habituelle de prendre entre 0 et 1 pour une question de lisibilite.

Pour le projet, il est assez subjectif de définir le coût colorimétrique pour passer d'un pixel à un autre. La norme sRGB (ou RVB) n'est pas représentative de la colorimétrie par rapport à l'œil humain. C'est à dire si on prend les valeurs RGB, une distance euclidienne identique entre pixel 1 et pixel 2 ou pixel 3 et pixel 4 peuvent avoir une valeur colorimétrique différente bien que la distance soit la même.

Dans mon projet, on utilise donc l'espace Cie L*a*b* pour décrire les couleurs, dont la distance euclidienne est applicable.
On va donc convertir les valeurs RGB en CIE XYZ (http://www.brucelindbloom.com/index.html?Eqn_RGB_to_XYZ.html) puis de CIE XYZ en CIE L*a*b*(https://fr.wikipedia.org/wiki/L*a*b*_CIE_1976).

Remarques sur les tests:

L'exécution d'exemples sur de grandes images peuvent prendre du temps:

```
grapheHolder.printChemin(grapheHolder.dijkstra(0,0,300,100));
```

Temps d'exécution : 687507 millisecondes.

Pour calculer l'algorithme de Dijkstra entre (0,0) et (300,100), le programme a mis 11 minutes environ pour finir. Cela comprend également l'affichage des valeurs de distance provisoires si elle est mise à jour et voir sur quelle arrête le programme est en train de travailler.

Pour accélérer le temps d'exécution, enlever:

```
System.out.println("Voisin : " + v.getNom() + ", Poids : " + poids);
```

et

```
System.out.println("Mise à jour de la distance pour le sommet " +
v.getNom());
```

des lignes 314 et 320 de Graphe.java.