ESIEE
PARIS

RAPPORT DE PROJET D'ALGORITHMIE

Théo PERESSE-GOURBIL et Manon HERMANN

Juin 2020

Contents

1 COCCINELLE ET PUCERONS

1.1 Réponse aux questions

1) Lors de son périple, la coccinelle a mangé 279 pucerons.
2) Son chemin sur la grille a été : (0,3)(1,2)(2,2)(3,1)(4,0)(5,0)(6,1)(7,0)

1.2 Raisonnement

Pour répondre au problème posé par la coccinelle, nous avions le choix parmi plusieurs algorithmes :

- Un algorithme glouton qui ne répondrait pas forcément correctement aux attentes;
- Un algorithme qui explore TOUTES les possibilités et chemins. Cette solution n'est pas envisageable car le temps de calcul serait alors factoriel, ce qui avec de grands tableaux n'est pas viable,
- Un algorithme en programmation dynamique, ce qui était demandé. En effet, la programmation dynamique a pour but de passer par une matrice M(l,c), ce qui correspond assez bien a notre problème puisque nous avons justement un tableau 2D qui peut être vu comme une matrice.

Nous avons donc fait un programme en programmation dynamique. Nous avons calculé une matrice M(l,c) qui avait les propriétés suivantes : chaque case correspond au nombre maximal de pucerons que la coccinelle a mangé en arrivant sur cette case. De ce fait, appelons Grille le tableau contenant les pucerons et M la matrice.

Lors de l'initialisation, la première ligne de M est parfaitement égale à la première ligne de la grille, puisque la coccinelle n'aura pas bougé.

Ensuite, il faut avancer d'une case (l augmente de 1). Sachant que l'on ne peut se déplacer que vers le Nord, le Nord-est ou le Nord-Ouest, en choisissant la solution qui donne accès au plus de pucerons.
On a donc :

```
M[l][c] = grille[l][c] + max3(M[l-1][c-1],M[l-1][c],M[l-1][c+1])
```

De proche en proche, on construit comme cela une matrice qui contient sur la dernière ligne le nombre total et maximal de pucerons que la coccinelle peut manger en arrivant sur cette case.
Afin d'obtenir le nombre maximal de pucerons mangés, il suffit de trouver le maximum de la dernière ligne de la matrice.
Pour connaître le chemin optimal, il faut parcourir la matrice M à l'envers. On part du plus grand de la dernière ligne, et on descend en allant soit au Sud, Sud-Est ou Sud-Ouest, tout en choisissant à chaque fois la case qui rapporte le plus de pucerons.
Attention toutefois à ne pas sortir de la grille, il faut donc vérifier la position sur la grille en permanence.

1.3 Explication du code

Dans cette section, nous expliquerons chaque fonction et son rôle, ainsi que des points sensibles de la fonction si besoin, dans l'ordre d'apparition dans le code source java.

1.3.1 Main

La fonction main est celle qui regroupe toutes les étapes du programme. En effet, elle sert juste à appeler correctement et dans l'ordre toutes les fonctions nécessaires dans l'algorithme, d'initialiser la grille de départ, gérer l'affichage des messages et mettre fin au programme.

1.3.2 calculGrille

Cette fonction permet la création de la matrice M selon la construction expliquée en 1.2. Nous initialisation une nouvelle matrice qui a la même taille que la grille, puis nous la remplissons, tout en faisant attention à ne pas dépasser.

1.3.3 printGrille

Cette fonction permet simplement d'afficher un tableau 2D.

1.3.4 max3 et max2

Ces deux fonctions similaires renvoient simplement le maximum de leur paramètres. Ces fonctions on été créées afin de ne pas surcharger le code avec des Math.max(x,Math.max(y,z));

1.3.5 maxLigne

Cette fonction calcule et revoie le maximum de la ligne passée en paramètre sous la forme d'un tableau à 2 cases : la première valeur correspond à la valeur maximum de la ligne et la seconde à son indice sur la ligne. La fonction ajoute en bas de la liste les coordonnées de la plus grande case de la matrice, puis remonte et se remplit progressivement, jusqu'à arriver au haut. Au final, on obtient dans l'ordre le chemin avec chaque coordonnées.

1.3.6 getTotal

La fonction getTotal est simple et permet de renvoyer le maximum de pucerons mangés en appelant maxLigne.

1.3.7 getPath

Sûrement la fonction la plus compliquée du programme, cette fonction permet de retracer le chemin de la coccinelle. Cette fonction part de la fin de M et remplit un tableau 2D en commençant par la fin. Cette méthode peut sembler complexe à mettre en oeuvre, mais une fois bien programmée, elle permet d'accéder rapidement à n'importe quelle case de la matrice ou de la grille.

1.3.8 getFirstCase et getLastCase

Ces deux fonctions permettent d'obtenir les coordonnées de la première et de la dernière case sur laquelle la coccinelle s'est rendue.

1.3.9 printCase et printPath

Ces deux fonctions d'affichage servent juste à afficher ce qui leur est demandées.

2 REDIMENSIONNEMENT D'IMAGES

Le problème posé était le suivant : comment redimentionner des images sans perdre l'information importante et sans que l'image ne subisse de distorsions. En effet, si on "rogne" une image sur les bords ou aléatoirement, il y a un risque de perdre de l'information. De plus, la compression sans perte d'information entraîne souvent des distorsions sur l'image.
Le but de ce programme est donc d'identifier les pixels importants, puis, grâce à la programmation dynamique, enlever les pixels les moins importants.

2.1 Raisonnement pour la résolution de problème

Le processus utilisé est celui du SeamCarving. Le SeamCarving se décompose en plusieurs étapes :

- Calcul selon un critère choisis préalablement sur chaque pixel afin de déterminer les pixels les moins importants. Nous pouvons utiliser la densité, l'entropie, la luminosité... Pour ce programme, nous avons décidé d'utiliser la méthode de Sobel, qui consiste en le calcul du gradient de la luminosité de chaque pixel en fonction des pixels qui l'entoure. Cela permet de bien définir les contours car selon nous, le plus important dans une image sont les variations brusques de couleurs. En effet, une variation brusque (donc un gradient élevé) correspond généralement à un changement de surface sur une image, un changement de milieu. C'est ce changement qui contient toute l'information. Les critères cités ci-dessus ne nous satisfaisaient pas. En effet, si l'on prend l'exemple de la luminosité, et que l'on applique le SeamCarving sur une image disons blanche avec un motif complexe noir au centre, lors du calcul de la luminosité, le noir, qui a une valeur RGB de 0, sera alors tout de suite supprimé. Or l'information résidait dans le motif et non dans le fond blanc.

- Par la suite, il faut calculer des chemins à travers l'image ayant le plus faible coût. C'est ici que la programmation dynamique intervient. En effet, un chemin à travers une image correspond exactement à l'inverse du problème de la coccinelle. Nous voulons "manger" le moins possible ici. Il suffit ensuite de recomposer une nouvelle image avec une ligne ou une colonne en moins, puis de répéter l'opération autant de fois que nécessaire.

Nous avons donc créer un constructeur qui dépend de l'état actuel de l'image. Ce constructeur a comme attributs des images et des valeurs. Les fonctions créées sont appelées sur cet objet, et ne concernent qu'une itération : enlever une ligne ou une colonne. Par la suite, dans les main, nous n'avons qu'à boucler le nombre voulu en créant à chaque fois un nouvel objet avec l'image créée par l'ancien constructeur, puis enlever une ligne ou une colonne.

La commande de compilation et d'exécution du programme est la suivante :

```
$ javac SeamCarving.java
$ java SeamCarving "path_to_image" "%x" "%y"
```

2.2 Explication du code

2.2.1 Les classes importées

Pour réaliser ce projet, nous avons eu besoin d'utiliser différentes classes, notament pour les images :

- La classe *BufferedImage* : Cette classe permet de récupérer des informations essentielles sur une image donnée comme la hauteur, la largeur, la valeur RGB de chaque pixel...
- La classe *ImageIO* : La classe ImageIO permet d'ouvrir, de lire, et d'enregistrer des images au format BufferedImage. Elle fait le lien entre le type File (inputStream) et BufferedImage grâce à la méthode read.
- La classe *File* : La classe File permet d'avoir accès à des inputStream, d'écrire et de lire des fichiers de tous types dans les fichiers de l'ordinateur. Cette classe nous a servi à ouvrir l'image souhaitée et à créer la nouvelle.
- La classe *Math* : Math est une classe comportant des méthodes qui réalisent des opérations mathématiques. Nous nous en sommes servi dans les fonctions de minimum et de racine carré.

2.2.2 Les attributs

Afin de réaliser notre algorithme, nous avons dans un premier temps créé une classe et un constructeur SeamCarving. Ce constructeur permettait d'initialiser facilement les attributs. Chaque objet de cette classe était principalement représenté par :

- Deux attributs du type BufferedImage, une input, qui était l'image avant transformation, et une output qui était l'image après transformation;
- Deux entiers, width et height, qui représentaient respectivement la largeur et la hauteur de l'input, en pixel;
- Une matrice RGBimage qui correspond a une matrice de même taille que l'image d'entrée, dont chaque case correspond à la valeur au format RGB du pixel correspondant;
- Une matrice EnergyImage qui correspond à une matrice de même taille que l'image d'entrée, dont chaque case correspond à la valeur d'énergie du pixel correspondant, en suivant la méthode de Sobel, c'est à dire par le gradient.
- Deux matrices Mx et My, de même taille que l'image, dont chaque case correspond au minimum d'énergie parcouru en étant positionnée sur cette case. Mx a son minimum en haut et son maximum en bas, tandis que My a son minimum à gauche et son maximum à droite.
- Deux tableaux, pathX et pathY, à deux dimensions, correspondant au chemin le plus court, avec le moins d'énergie, parcourant la matrice Mx et My. La première case représente la coordonnée en x et la seconde en y.

2.2.3 Constructeur *SeamCarving*

Le constructeur de la classe prend en paramètre une image de type BufferedImage et une chaîne de caractère. Il initialise tous les attributs. Le mode, représenté par la chaîne *mode* permet de choisir entre 3 modes d'actions, afin de ne pas effectuer de calculs inutiles :

- Le mode *0*: Ce mode permet d'initialiser la première fois les attributs, sans effectuer de calculs;
- Le mode x: Ce mode permet d'initialiser les attributs comme précédemment, mais en plus de lancer le calcul de Mx et pathX

- Le mode y: Comme pour le mode x, ce mode permet de lancer les calculs de My et pathY

Les différents modes permettent de gagner du temps sur les calculs. Lorsque l'on veut supprimer des pixels en x, inutile de calculer My et pathY, qui sont les deux fonctions les plus gourmandes en temps.

2.2.4 getrgbImage

Cette fonction permet d'obtenir la couleur en RGB de chaque pixel de l'image. Elle transforme l'image d'entrée en une matrice rgbImage qui pourra ensuite être traitée dans le reste du code.

2.2.5 getLuminance

La fonction getLuminance permet d'obtenir un nombre représentant la quantité de luminance en chaque pixel. La fonction getRGB de la classe BufferedImage permet d'obtenir au format RGB la valeur RGB d'un pixel. Sous le format RGB, les 8 premiers bits sont pour le bleu, les 8 suivants pour le vert et les 8 derniers pour le rouge.
La formule suivante permet d'obtenir le niveau de gris correspondant :

```
gris = (299*r + 587*g + 114*b)/1000;
```

2.2.6 gradient

Cette fonction utilise la méthode de Sobel. Pour chaque pixel, on regarde les pixels aux alentours (x+1, x-1, y+1,y-1) et on calcule la luminance de chacun d'entre eux grâce à la fonction getLuminance. On calcul ensuite le gradient en x et en y, pour ensuite en calculer la norme. Cette norme correspond au gradient du pixel(x,y) entré dans la fonction.

2.2.7 setEnergy

La fonction setEnergy permet d'éditer la matrice energyImage contenant le valeur du gradient pour chacun de ses pixels. On utilisera par la suite cette matrice pour déterminer les pixels avec le moins d'importance.

2.2.8 calculMx et calculMy

Dans ces deux fonctions, on générera deux matrices Mx et My respectivement qui serviront à identifier le chemin de coût minimum en x et en y, afin de les retirer de l'image.

2.2.9 getPathX et getPathY

Les deux fonctions servent à calculer le chemin avec le coût le plus faible. Pour getPathX, on recherche l'énergie la plus petite de la dernière ligne. Puis on regarde, pour les pixels accessibles, celui qui est le plus petit. On parcours ainsi la matrice Mx de droite à gauche. Toutes les coordonnées sont entrées dans la variable pathX qui constitue le chemin de coût minimum. On fait de même dans getPathY, mais on parcours la matrice My de bas en haut.

2.2.10 minColonne et minLigne

Ces deux fonctions servent respectivement à calculer le minimum de la dernière colonne et le minimum de la dernière ligne. Ces deux fonctions servent au commencement de getPath.

2.2.11 removeX et removeY

Dans ces fonctions, on crée une nouvelle image que l'on va compléter sans les pixels appartenant à Path, cela revient à réduire de 1 la taille horizontale ou verticale de l'image d'origine. Pour ce faire, on recopie chaque pixel jusqu'à arriver à celui qu'il faut enlever, on saute ce pixel pour ensuite continuer à recopier les suivants décalé de 1. La seule différence entre removeX et removeY est dans le sens de parcours des images.

2.2.12 getPourcentage

La fonction getPourcentage permet de convertir une chaîne de caractère saisie par l'utilisateur lors de l'exécution du programme en un Integer, tout en vérifiant le format.

2.2.13 openImg

Cette fonction sert de test pour ouvrir l'image au tout début du programme. Elle transforme un fichier, de type File, en une image exploitable par notre programme, de type BufferedImage.

2.2.14 main

La fonction main est celle qui regroupe toutes les étapes du programme. Elle vérifie tout d'abord que la commande est juste : il faut 3 arguments (nom de l'image, pourcentage de réduction horizontale, pourcentage de réduction verticale). Ensuite, elle sert juste à appeler correctement et dans l'ordre toutes les fonctions nécessaires dans l'algorithme. C'est aussi elle qui définie le nombre de fois que des lignes et colonnes doivent être supprimées pour obtenir la bonne réduction.

2.3 Problèmes rencontrés

Lors de l'élaboration de ce projet, nous avons fait face à plusieurs problèmes :

- Trouver le moyen d'éditer les images
  L'édition d'image était nouveau pour nous, nous avons donc du découvrir une nouvelle partie de Java, avec de nouvelles classes et fonctions. Nous avons mis un certain temps à parcourir la documentation officielle Oracle et trouver les fonctions dont nous avions besoin.

- Gradient de luminance :
  Afin de trouver le bon paramètre, nous avons du chercher longtemps car comme nous l'expliquons plus haut, certains critères ne sont pas adaptés à cette méthode. La méthode de Sobel sur le gradient fonctionne particulièrement bien.

- Être cohérent entre x et y :
  En informatique, le pixel de coordonnée (0,0) se situe en haut à gauche. De ce fait, il est compliqué de se représenter des tableaux inversés, la nomenclature devient alors très importante. Une erreur sur x ou sur y peut conduire à des résultats assez exotiques.

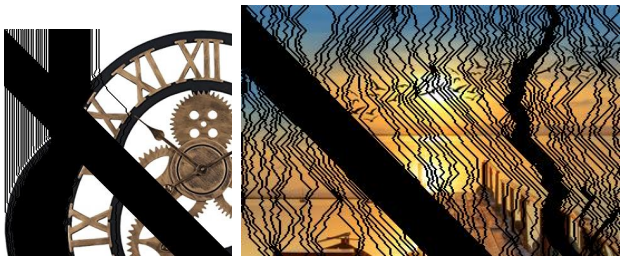

Figure 1: cafouillage entre les coordonnées x et y

- Raisonnement pour supprimer plusieurs fois une ligne/colonne :
  Un problème s'est posé à la fin de notre projet. En effet, nous avions tout fait pour supprimer et identifier une ligne ou colonne à supprimer. Nous avons donc trouvé la solution : d'instancier l'objet le nombre de fois nécessaire et faire une boucle pseudo-récursive.

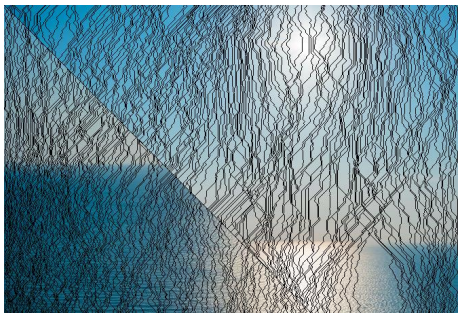

Figure 2: problème décalage

Afin d'éviter tout soucis au niveau organisation dans l'avancement du projet, plusieurs outils nous ont aidés :

- Répertoire GitHub : `https://github.com/CappiLucky/Peresse_Hermann`
- Fonction télétype de Atom : pour pouvoir travailler en simultané sur le même document
- OverLeaf : Client LaTeX pour le rapport permettant de travailler en simultané sur le rapport

3 Sources

- https://www.developpez.net/forums/d418823/general-developpement/algorithme-mathematiques/contribuez/image-seam-carving/
- https://fr.wikipedia.org/wiki/Seam_carving
- https://docs.oracle.com/javase/7/docs/api/java/awt/image/BufferedImage.html
- https://docs.oracle.com/javase/7/docs/api/javax/imageio/ImageIO.html
- https://docs.oracle.com/javase/7/docs/api/java/io/File.html
- https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html
- https://docs.oracle.com/javase/tutorial/2d/overview/images.html
- https://en.wikipedia.org/wiki/Sobel_operator