

Rapport

Exercice 1

Compilation et execution du programme

```
javac Coccinelle.java  
java Coccinelle
```

Réponses aux questions

1. La coccinelle a mangé 279 pucerons.
2. Elle a suivi le chemin (0,3)(1,2)(2,2)(3,1)(4,0)(5,0)(6,1)(7,0)
Case d'atterrisage = (0,3).
Case de l'interview = (7,0).

Explications du programme

On va tout d'abord créer un nouveau tableau avec le même nombre de lignes et de colonnes que celui donné dans l'énoncé. Pour cela on créer la méthode tri qui prend en entrée, 2 tableau. Ici nos 2 tableaux sont le tableau donné par l'énoncé (tab) et le nouveau tableau (tabtri). On va ensuite copier les valeurs de tab dans tabtri, on pourra ensuite le trier. On définit une variable permettant de stocker la position de la case voisine ayant le plus grand nombre de pucerons autrement dit en : (x-1,y+1) ; (x,y+1) ; (x+1,y+1) .

Une fois la case trouvée on ajoute le total de pucerons accumulé jusqu'à maintenant à la valeur de cette case dans tabtri. Une fois fait pour la première ligne on change de ligne et etc... On obtient alors notre tableau trié avec, sur la dernière ligne, le nombre potentiel de pucerons ingéré maximum (représentant l'importance ou non du trajet: plus le nombre est élevé et plus le chemin finissant sur cette case est rentable pour la coccinelle). On cherche ensuite la case ayant la valeur la plus grande sur la dernière ligne, cette case est nécessairement la case d'arrivée de notre coccinelle ! (On pourra ensuite parcourir le chemin inverse depuis cette case pour connaître le trajet parcourus)

Cela va être le rôle de la méthode `Parcours()` qui va comme son nom l'indique retrouver le parcours le plus rentable pour la coccinelle grâce au tableau trié. Cette méthode prendra en entrée une liste vide qui stockera les différentes localisations des cases du chemin parcouru par la coccinelle, le tableau trié et la position de la dernière feuille ou la coccinelle doit se trouver. À chaque itération le programme va regarder dans le tableau trier la valeur des cases précédente et va choisir celle ayant la plus grosse valeur. Il renverra la position de cette case qui sera stocké dans la liste. La récursivité de la fonction permet d'avancer de case en case jusqu'à trouver la case de départ de la coccinelle. On obtiendra une liste avec les différentes positions des cases empruntés par la coccinelle, ainsi que le point de départ et le point d'arrivée.

Exercice 2

Compilation et execution du programme

```
javac SeamCarving.java  
java SeamCarving [File] [Horizontal %] [Vertical %]
```

Explications du programme

La fonction Main a pour but de vérifier les arguments entrés par l'utilisateur, en particulier concernant les pourcentages de réduction de l'image, il faut impérativement qu'il soit possible de les transformer en float afin de réaliser la division permettant de connaître le nombre de *lignes* à retirer de l'image. Il faut également que ces pourcentages ne dépassent pas 100 pour des raisons évidentes.

Afin d'enclencher le processus de réduction d'image par la méthode de Seam Carving on crée une nouvelle instance process de la classe SeamCarving ce qui fait appelle au constructeur par défaut qui s'occupera d'appeler les procédures au fur et à mesure.

Afin d'éclaircir les codes nous utilisons 4 attributs dans la classe SeamCarving, la largeur et la hauteur de l'image en pixels, un tableau de 2 dimensions contenant l'énergie de chaque pixels et enfin l'image tampon (BufferedImage) facilement manipulable par Java. Ces attributs seront actualisés au fur et à mesure de l'avancement du programme.

Voici le détail algorithmique des procédures appelés par le constructeur:

- Initialisation
 - Génération de l'image tampon à partir du fichier
 - Récupération de la largeur et de la hauteur de l'image
 - Récupération du nombre de *lignes* / *colonnes* (horizontales / verticales) à retirer en fonction des pourcentages et de la valeur réelle des (largeurs / hauteurs)
 - Récupération de la valeur RGB de chaque pixels dans un tableau à 3 dimensions
 - Génération d'un tableau à 2 dimensions contenant l'énergie de chaque pixels en fonction des pixels voisins (grâce au tableau RGB à 3D)
- Récurrence
 - Suppression des lignes verticales (boucle incrémenté en fonction du nombre de *colonnes* à retirer) [voir détail plus bas]
 - Suppression des lignes horizontales (boucle incrémenté en fonction du nombre de *lignes* à retirer) [voir détail plus bas]
- Finalisation
 - Récupération du nom du fichier (et non le chemin associé)
 - Sauvegarde de l'image tampon dans un fichier

Voici le détail algorithmique des procédures associé à la récurrence:

- Suppression des *colonnes* (boucle incrémenté en fonction du nombre de *colonnes* à retirer)
 - Récupération dans une liste des pixels constituant le trajet vertical possédant l'énergie totale la plus faible.
 - On se place à chaque début de colonnes (boucle incrémenté en fonction de la largeur de l'image)
 - on regarde les pixels voisins sur la ligne inférieure autrement dit en : $(x-1,y+1)$ - $(x,y+1)$ - $(x+1,y+1)$
 - On retient le pixel possédant l'énergie la plus faible
 - On retient la liste possédant l'énergie totale la plus faible
 - Suppression de la *colonne* associée à la liste de pixel
 - Création d'une nouvelle image tampon avec une largeur réduite de 1
 - Remplissage de la nouvelle image tampon
 - On recopie un à un les pixels de l'ancienne image tampon
 - On ne recopie pas dans la nouvelle image tampon les pixels aux coordonnées sauvegardé dans la liste
 - On se décale d'une position une fois que le pixel de la ligne a été ignoré
 - On remplace l'ancienne image tampon par la nouvelle
 - On réduit la valeur de la largeur de l'image de 1
 - On régénère le tableau d'énergie associé à la nouvelle image tampon
- Suppression des *lignes* (boucle incrémenté en fonction du nombre de *lignes* à retirer)
 - Confère le détail de la suppression des *lignes* verticales à ceci près que l'on travaille sur les *lignes* et non les *colonnes*

En fin de page nous avons commenté trois procédures :

- `extractHorizontal()`
- `extractVertical()`
- `extractEnergy()`

Ces procédures permettent d'exporter l'énergie de l'image à son état d'initialisation afin d'avoir un aperçu visuel de ce qu'est le tableau d'énergie ainsi qu'une *ligne* et une *colonne*. Confère les images ci-dessous.

Exemple de sortie :

```
Horizontal process : [121/121] / (^-^) \
Vertical process : [383/383] / (^-^) \
Process done ! \ (^-^) /
```

Exemples de redimensionnement :

Image d'origine [chaton.png]:



Crédit Photo : iStock/Annala

Image redimensionnée [chaton_resized_20_50.png] :



Crédit Photo : iStock/Annala

Image d'origine [chat.png] :



Image redimensionnée [chat_resized_20_50.png] :



Visualisation de l'énergie:



Visualisation d'une *ligne* horizontale:



Visualisation d'une *ligne* verticale:

