Nous allons dans ce TP programmer l'algorithme *Seam Carving* ou de recadrage intelligent développé par S. Avidan et A. Shamir.

Si on veut recadrer une image bitmap (disons en largueur) on peut penser à plusieurs méthodes :

- On supprime les parties de l'image le plus à droite (ou à gauche ou les deux), mais on peut perdre des parties essentielles de l'image.
- On a recours à des calculs d'extrapolation mais on risque d'aboutir à une déformation de l'image.

Le principe de l'algorithme de Seam Carving est de supprimer les pixels les « moins utiles ».

Nous travaillerons dans la suite (pour un soucis de simplicité) avec des images en noir et blanc. On supposera que les images sont codées par une matrice d'entiers compris entre 0 et 255. Chaque case de la matrice correspondant à un pixel de l'image, le ton 0 correspond au noir et le ton 255 au blanc.

Si `m` est une matrice on notera `h` la hauteur de l'image que l'on peut obtenir par `Array.length m` et `l` la largueur de l'image que l'on peut obtenir par `Array.length m.(0)`.

Rappelons aussi que l'on peut créer une nouvelle matrice avec `n` lignes et `p` colonnes contenant la valeur `x` par `Array.make_matrix n m x`.

À chaque pixel d'une matrice `m` on associe son *énergie* qui code son importance. Plus l'énergie est grande plus le pixel est important. On peut penser à plusieurs définitions pour l'énergie. Pour ce TP on pose pour l'énergie du pixel `(i,j)` :

$$e_{i,j} = \sqrt{(m_{i,j-1} - m_{i,j+1})^2 + (m_{i-1,j} - m_{i+1,j})^2}$$

1) a) Écrire une fonction `energie : int array array -> float array array` telle que `energie mat` renvoie la matrice des énergies des pixels. On affectera aux cases des « bords » de la matrice le maximum des valeurs de la matrice afin de ne pas modifier les pixels sur les bords de l'image.

   b) Écrire une fonction `retireCaseTableau : 'a array -> int -> a' array` telle que `retireCaseTableau t i` renvoie le tableau obtenu en enlevant la i-ème case du tableau `t`.

c) Écrire une fonction `retireCaseMatrice : 'a array array -> int -> int -> unit` telle que `retireCaseMatrice mat i j` retire la case `mat.(i).(j)` de la matrice `mat`.

2) a) Écrire une fonction `supprime1 : int array array -> unit` telle que `supprime1 mat` supprime sur chaque ligne de la matrice `mat` le pixel de plus basse énergie. En déduire une fonction `supprime : int array array -> int -> unit` telle que `supprime mat i` applique i fois la fonction précédente.

b) Tester votre fonction pour enlever 50 colonnes de l'image `tower.pgm`. On utilisera les fonctions `imageToMatrice` et `matriceToImage`. Que penser du résultat ? *Le calcul est assez long.*

3) Pour éviter le problème aperçu précédemment, on va chercher à enlever des « coutures verticales ». Précisément, on appelle *couture verticale* dans une matrice à h lignes et l colonnes, la donnée d'un h-uplet de couples $(i, x(i)))_{0\leq i\leq h-1}$ tels que pour tout i non nul $|x(i) - x(i-1)| \leq 1$. On appelle alors énergie de la couture la somme des énergies des pixels de la couture. C'est-à-dire le réel : $\sum_{i=0}^{h-1} e_{x(i),i}$.

a) À l'aide d'un algorithme utilisant la programmation dynamique [1] écrire une fonction `tableauCouture : float array array -> float array array` telle que `tableauCouture ener` (`ener` étant la matrice des énergies calculée ci-dessus) renvoie la matrice `p` telle que `p.(i).(j)` soit l'énergie d'une couture d'énergie minimale parmi les coutures allant du haut de l'image au pixel `(i,j)`. On commencera par déterminer une formule de récurrence définissant `p.(i).(j)` en fonction des `p.(i-1).(k)`.

b) En déduire une fonction `couture : float array array -> int list` telle que `couture ener` renvoie la liste des abscisses des points d'une couture d'énergie minimale. On sera amené à modifier la fonction ci-dessus de sorte à pouvoir reconstituer la couture.

c) Écrire une fonction `seam1 : int array array -> unit` telle que `seam1 mat` supprime de la matrice une couture de plus basse énergie. En déduire une fonction `seam : int array array -> int -> unit` telle que `seam mat i` applique i fois la fonction précédente.

d) Tester votre fonction en utilisant l'image `tower.pgm`. On pourra enlever 50 colonnes. *Le calcul est assez long.*

4) Reprendre la question 3) mais en récursif si vous l'aviez traitée en itératif et réciproquement.

1. on choisira entre un algorithme itératif type « bottom-up » et un algorithme récursif avec mémoïzation