EXERCICE 3 (4 points)

En informatique, chaque pixel d'une image numérique est affiché sur un écran standard par synthèse additive du rouge, du vert et du bleu : c'est le système colorimétrique RVB. Chacune des trois composantes RVB d'un pixel est stockée sur un octet.

Pour représenter une image, un fichier au format *Windows bitmap* (BMP) contient un entête, suivi des valeurs des composantes RVB de chacun des pixels, écrites les unes à la suite des autres.

1. On considère une image de hauteur 1000 pixels et de largeur 1500 pixels.
   Combien de mégaoctets faut-il pour représenter l'ensemble des pixels de cette image en omettant l'entête ?

2. En Python, les images sont généralement représentées à l'aide de tableaux de tableaux.
   - les composantes RVB d'un pixel sont stockées dans un `tuple` de trois entiers compris entre 0 et 255 ;
   - pour chaque ligne de l'image, les composantes RVB des pixels sont stockées dans un tableau ;
   - l'image est alors représentée par un tableau `img` contenant tous les tableaux précédents ; par convention, l'image vide est représentée par le tableau contenant un tableau vide `[[]]`.

   Ainsi, si l'image `img` a une hauteur de 1000 pixels et largeur de 1500 pixels,

```
>>> len(img)
1000
>>> len(img[0])
1500
```

   La position de chaque pixel dans une image `img` est repérée par un couple d'entiers `(i, j)` qui sont respectivement les indices de ligne et de colonne du pixel dans `img`.

   Encadrer `i` et `j` lorsque l'image `img` a pour hauteur `n` et largeur `m`.

Lorsque `i` et `j` vérifient ces encadrements, on dira qu'ils sont **compatibles** avec la taille de l'image.

On s'intéresse dans la suite de cet exercice à l'algorithme de recadrage intelligent (*seam carving*), qui permet de réduire la largeur d'une image tout en conservant ses principales caractéristiques graphiques.

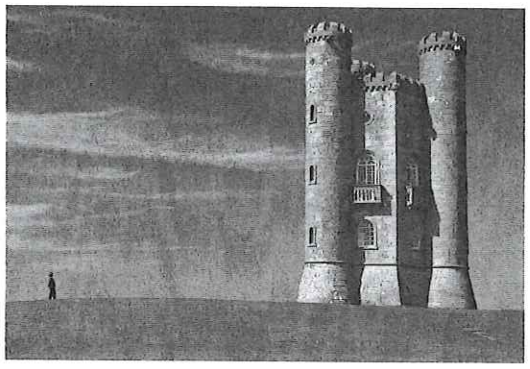
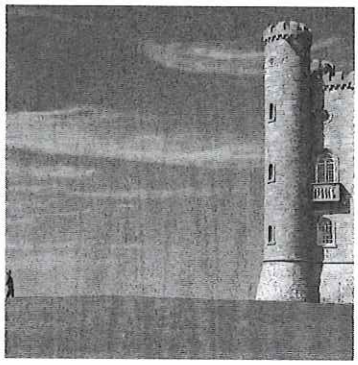
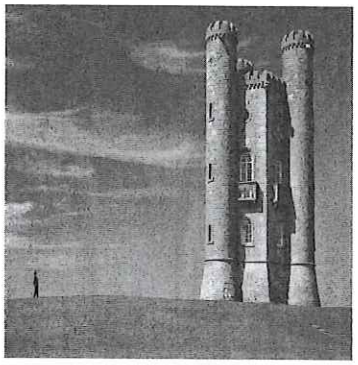


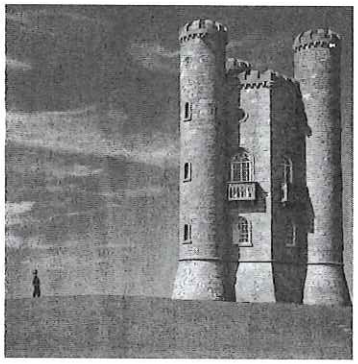
Image dont on souhaite réduire la largeur



Découpe de l'image
une partie du château n'est plus visible



Redimensionnement
le château est déformé

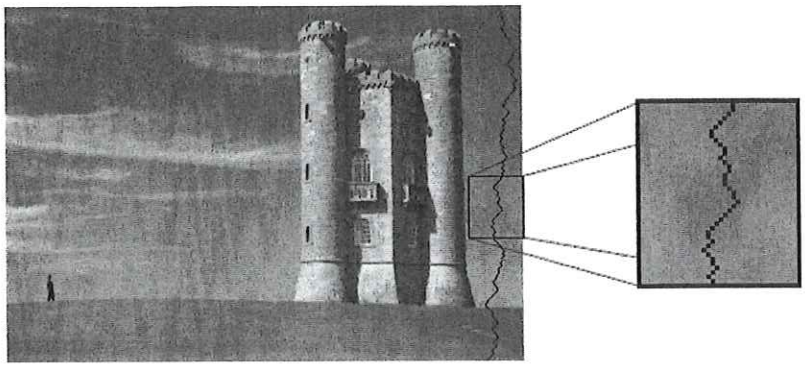


Seam carving
les aspects des principaux éléments graphiques sont conservés

Source des illustrations : https://en.wikipedia.org/wiki/Seam_carving

L'algorithme de recadrage intelligent s'appuie sur la suppression de pixels « bien choisis » dans l'image. Pour déterminer ces pixels, il recherche ce que l'on appellera la « couture de moindre énergie ».

Une couture est une suite de pixels adjacents allant du haut au bas de l'image. On associe à chaque pixel de l'image un nombre positif appelé énergie du pixel. L'énergie d'une couture est alors la somme des énergies des pixels qui la composent.



Une couture et un détail de cette couture.

L'algorithme de recadrage intelligent d'une image de dimensions `n` × `m` (hauteur `n`, largeur `m`) détermine alors la couture de plus basse énergie et supprime de l'image tous les pixels présents dans la couture. L'image résultante est de dimensions `n` × `(m - 1)` (hauteur `n`, largeur `m - 1`). On recommence le procédé jusqu'à obtenir la largeur souhaitée.

3. On suppose que l'on dispose d'une fonction `energie(img, i, j)`, qui prend en paramètres une image `img`, un indice de ligne `i`, un indice de colonne `j` et renvoie l'énergie du pixel à la position `(i, j)` de l'image.
On décide de calculer au préalable les énergies des différents pixels de l'image. La fonction `calcule_tab_energie` prend comme paramètre une image `img` et renvoie le tableau de tableaux des énergies correspondantes. Ainsi `tab_energie[i][j]` contiendra à la fin de l'exécution l'énergie du pixel à la position `(i, j)`.

```
def calcule_tab_energie(img):
    n = len(img)
    m = len(img[0])
    tab_energie = []
    for i in range(n):
        ligne = []
        for j in range(m):
            e = energie(img, i, j)
            ligne.append(e)
        tab_energie.append(ligne)
    return tab_energie
```

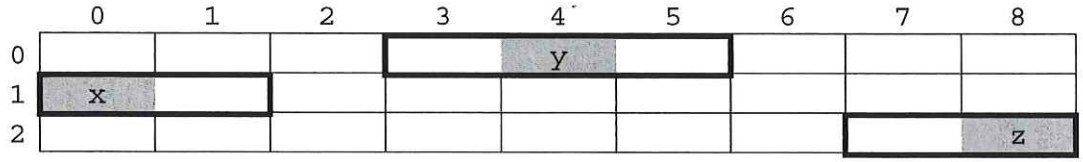
a. Quel est le type de la variable `tab_energie` renvoyée par la fonction `calcule_tab_energie(img)` ?

b. Exprimer en fonction de `n` et de `m` le nombre d'appels à la fonction `energie`.

4. On représente en Python une couture à l'aide d'un tableau de couples d'indices `(i, j)` compatibles avec la taille de l'image. Chaque couple `(i, j)` repère la position dans l'image d'un pixel de la couture.
On rappelle que l'énergie d'une couture est la somme des énergies des pixels qui la composent.

Écrire une fonction `calcule_energie(couture, tab_energie)`, qui prend en paramètres un tableau `couture` représentant une couture ainsi que le tableau des énergies `tab_energie` renvoyé par la fonction `calcule_tab_energie`. Cette fonction doit renvoyer l'énergie de la couture passée en argument.

5. Écrire la fonction `indices_proches(m, i, j)`, qui prend en paramètres `m` le nombre de colonnes de l'image, `i` un indice de ligne, `j` un indice de colonne, et renvoie parmi les positions des pixels `(i,j-1)`, `(i,j)` et `(i,j+1)` celles qui sont compatibles avec les dimensions de l'image.
On ne tiendra pas compte de l'ordre des éléments de la liste renvoyée.
Par exemple, on a repéré dans l'image de dimension 3×9 ci-dessous trois pixels `x`, `y`, `z`.




```
>>> indices_proches(9, 1, 0) # pixels voisins de x
[(1,0), (1,1)]
>>> indices_proches(9, 0, 4) # pixels voisins de y
[(0,3), (0,4), (0,5)]
>>> indices_proches(9, 2, 8) # pixels voisins de z
[(2,7), (2,8)]
```

6. On appellera couture d'indice `j` une couture dont la position du premier pixel est `(0, j)`. Afin de déterminer une couture de basse énergie parmi toutes les coutures d'indice `j`, on met en œuvre une stratégie de type glouton.
Pour cela, on construit la couture du haut vers le bas en sélectionnant à chaque étape le pixel d'énergie minimale parmi ceux situés immédiatement en bas à gauche, immédiatement en bas ou immédiatement en bas à droite lorsque cela est possible.

Par exemple, on donne ci-dessous un tableau `tab_energie` d'une image de dimension `4 × 3`. Les pixels de la couture d'indice `j = 1` y ont été grisés.

	0	1	2
0	3	4	2
1	4	5	6
2	3	2	1
3	7	8	6

Le premier pixel de la couture a pour position `(0, 1)`
Le 2e pixel sera choisi parmi `[(1,0), (1,1), (1,2)]`
Le 3e pixel sera choisi parmi `[(2,0), (2,1)]`
Le 4e pixel sera choisi parmi `[(3,0), (3,1), (3,2)]`

La couture d'indice 1 est donc : `[(0,1), (1, 0), (2, 1), (3, 2)]`

La fonction `calcule_couture(j, tab_energie)`, ci-dessous a pour paramètres un entier `j` et le tableau des énergies `tab_energie`. Cette fonction doit renvoyer une liste de tuples représentant la couture d'indice `j` construite selon le procédé décrit précédemment.
On pourra utiliser la fonction `indices_proches`, ainsi que la fonction `min_energie(tab_energie, indices_pixels)`, qui prend en paramètres le tableau d'énergie `tab_energie` et la liste de positions de pixels `indices_pixels`. La fonction `min_energie` renvoie le tuple `(i,j)` de la position du pixel de moindre énergie.

Recopier et compléter le code de la fonction `calcule_couture` ci-dessous :

```
def calcule_couture(j, tab_energie) :
    n = len(tab_energie)      # hauteur de l'image
    m = len(tab_energie[0]) # largeur de l'image
    couture = []
    couture.append((0, j))    # premier pixel de la couture
    for i in range(1, n):     # i est l'indice de la ligne du
                              # prochain pixel dans la couture
        ...                   # plusieurs lignes possibles

    return couture
```

7. Le paramètre `j` de la fonction précédente peut prendre toutes les valeurs entières comprises entre `0` (inclus) et `m = len(tab_energie[0])` (exclu).
Écrire une fonction `meilleure_couture(tab_energie)`, qui renvoie une couture d'énergie minimale parmi celles obtenues à l'aide de la fonction `calcule_couture`.
L'algorithme consistera à parcourir tous les indices `j` et ne garde qu'une couture d'indice `j` qui soit d'énergie minimale. Pour cela, on pourra utiliser les fonctions `calcule_couture` et `calcule_energie` définies auparavant.