Diviser pour Regner

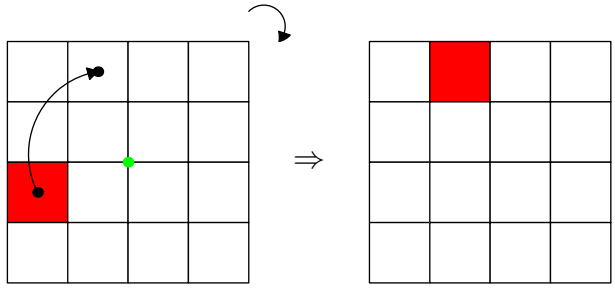
Faire tourner une image d'un quart de tour

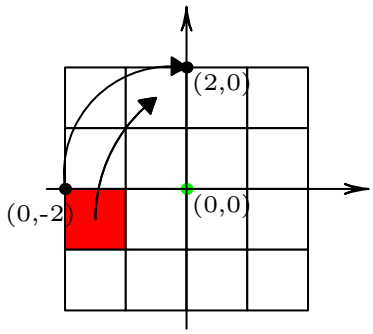
Le but est de montrer comment programmer une rotation d'un quart de tour d'une image d'une part avec une méthode itérative, d'autre part en mettant en oeuvre la méthode *diviser pour régner*. *D'après un document de Laurent Chenot.*

Coordonnées d'un pixel après un quart de tour

Une image est constituée de pixels. Un pixel est un carré unitaire défini par ses composantes (rouge, vert, bleu) et par la position du coin **supérieur gauche** dans le repère de l'écran. Pour comprendre l'algorithme itératif, il est important de savoir déterminer quelle place un pixel va prendre après avoir fait un quart de tour autour du centre de l'image.

Dans l'image ci-contre, voici le déplacement d'un pixel lorsqu'on effectue un **quart** de tour dans le sens horaire, autour du centre de l'image.
Nous nous placerons pour commencer dans un repère avec origine au **centre de l'image**.

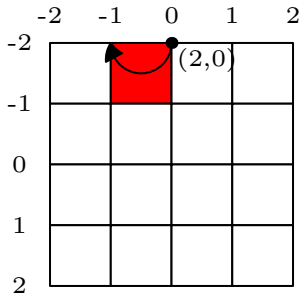




Si au lieu d'un pixel, on regarde tout d'abord un point. On voit par exemple que le point de coordonnées $(0, -2)$ passe en $(2, 0)$. Autre exemple : $(-2, 1) \longrightarrow (\ldots, \ldots)$.

Proposez une expression des coordonnées (i', j') après un quart de tour, en fonction des coordonnées initiales (i, j) d'un point de la grille :

$$(i', j') = \ldots\ldots\ldots$$



Cette formule est valable pour le quart de tour d'un point. Pour un pixel, il faut y apporter une modification afin de toujours donner les coordonnées du coin supérieur gauche !

Proposez une expression des coordonnées (i'', j'') après un quart de tour en fonction des coordonnées précédentes (i', j') du pixel :

$$(i'', j'') = \ldots\ldots\ldots\ldots$$

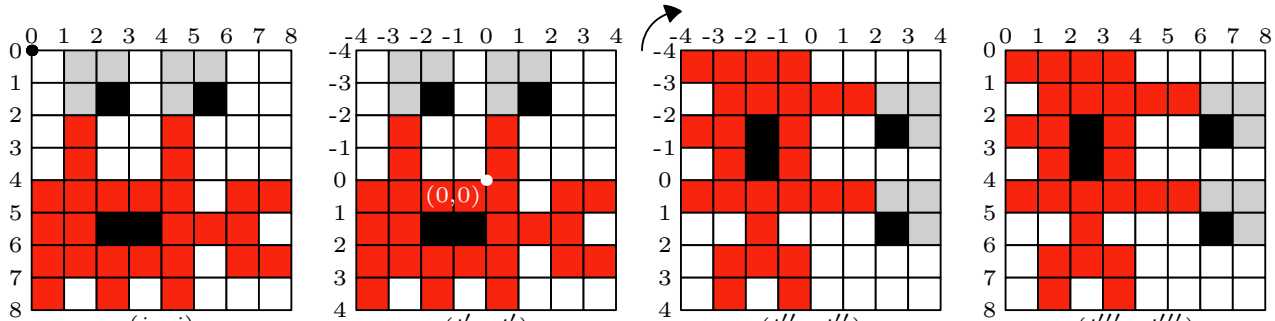
En déduire une expression des coordonnées (i'', j'') en fonction des coordonnées initiales (i, j) du pixel :

$$(i'', j'') = \ldots\ldots\ldots\ldots$$

Il reste une dernière difficulté à surmonter : l'origine du repère de l'écran n'est pas le centre de l'image comme dans la sitation précédente. L'origine est le coin supérieur gauche de l'écran. Une bonne façon de procéder est de changer de repère pour pouvoir utiliser le résultat précédent.

Activité :

1) Dans chaque cas, donner la position du pixel dans les grilles successives.
Exemple : $(2, 5) \longrightarrow (-2, 1) \longrightarrow (-2, -2) \longrightarrow (2, 2)$



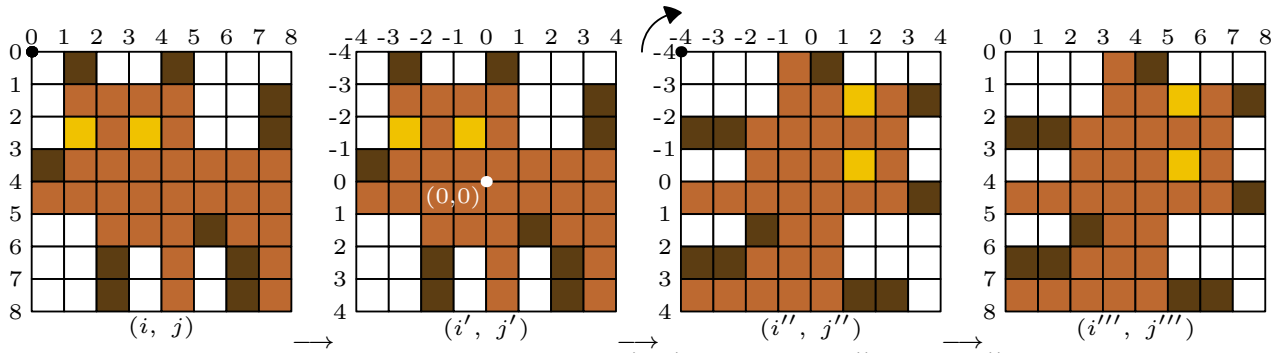
$(i,\ j)$	$(i',\ j')$	$(i'',\ j'')$	$(i''',\ j''')$
$(1, 2)$	$(\ ,\ )$	$(\ ,\ )$	$(\ ,\ )$
$(5, 1)$	$(\ ,\ )$	$(\ ,\ )$	$(\ ,\ )$
$(7, 5)$	$(\ ,\ )$	$(\ ,\ )$	$(\ ,\ )$
$(2, 7)$	$(\ ,\ )$	$(\ ,\ )$	$(\ ,\ )$

2) Proposez des expressions de changement de coordonnées en fonction des précédentes dans chaque cas :

$(i', j') =$ ..... $(i'', j'') =$ ..... $(i''', j''') =$ .....

3) En déduire l'expression des coordonnées finales en fonction des coordonnées initiales :
$(i''', j''') =$ .....
.....

4) Utilisez les expressions proposées sous les flèches pour obtenir les coordonnées des pixels suivants.



$(i,\ j)$	$(i',\ j')$	$(i'',\ j'')$	$(i''',\ j''')$
$(0, 3)$	$(\ ,\ )$	$(\ ,\ )$	$(\ ,\ )$
$(3, 2)$	$(\ ,\ )$	$(\ ,\ )$	$(\ ,\ )$
$(5, 5)$	$(\ ,\ )$	$(\ ,\ )$	$(\ ,\ )$
$(2, 5)$	$(\ ,\ )$	$(\ ,\ )$	$(\ ,\ )$

Vérifiez que les coordonnées finales correspondent avec le graphique et qu'elles correspondent à la formule trouvée à la question 3).

5) On souhaite faire tourner l'image *en place* c'est-à-dire sans en créer une copie. Lorsqu'on effectue un quart de tour d'un pixel, il est nécessaire de faire tourner 4 pixels simultanément. Exprimez les coordonnées de ces quatre pixels en fonction des coordonnées (i, j) et de $r_i = n - 1 - i$ et $r_j = n - 1 - j$:

pix0= (i,j) pix2 = ...
pix1 = ... pix3 = ...

Algorithme itératif de rotation d'une image d'un quart de tour.

- La fonction `echange2pix` échange *sur place* les pixels de coordonnées `ij0` et `ij1` dans `image`.

- La fonction `tourne4pix` effectue une permutation circulaire de 4 pixels dans `image`.

- La fonction `tourne` effectue un quart de tour par 4 pixels de tous les pixels de `image`.

On retrouve les coordonnées des 4 pixels de la question 5) ci-dessus !

```
def echange2pix(image,ij0,ij1):
    couleurs0 = image.getpixel(ij0)
    couleurs1 =image.getpixel(ij1)
    image.putpixel(ij0,couleurs1)
    image.putpixel(ij1,couleurs0)

def tourne4pix(image,ij0,ij1,ij2,ij3):
    echange2pix(image,ij0,ij1)
    echange2pix(image,ij1,ij2)
    echange2pix(image,ij2,ij3)

def tourne(image):
    assert (img.size[0] == img.size[1])
    n = img.size[0]
    for i in range(n//2):
        for j in range((n+1)//2):
            ri = (n-1) - i
            rj = (n-1) - j
            tourne4pix(image,(i,j),(rj,
   i),(ri,rj),(j,ri))
```

Pour tester ce code, ajoutez au début :

```
from PIL import Image
img = Image.open("joconde_1024.png")
import time
img.show()
img.size
```

Quels commentaires pouvez-vous faire ?

.....

.....

.....

Puis à la fin :

```
temps_origine = time.time()
tourne(img)
print(time.time()-temps_origine)
img.show()
```

Notez le temps de traitement obtenu :

.....

.....

Complexité

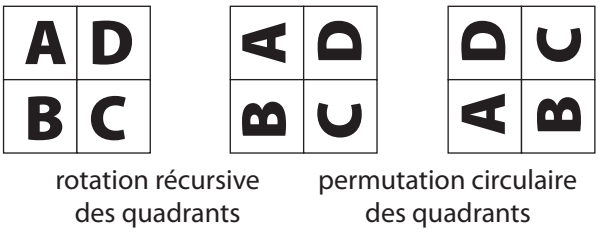
La compléxité de l'algorithme est $\mathcal{C}(n) =$ .....
Avec $\mathcal{O}(1)$ mémoire auxiliaire car c'est un traitement *en place* (espace mémoire constant).

Algorithme récursif de type diviser pour régner

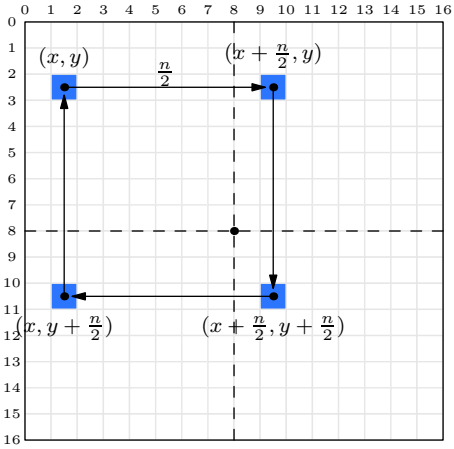
On suppose l'image carrée et de côté puissance de 2 pour simplifier la présentation. Principe : on découpe le carré en quatre quadrants, on fait tourner récursivement chaque quart, puis on opère une permutation circulaire des quadrants.

```python
def echangeQuadrant(image,ij0,ij1,n):
    x0,y0 = ij0
    x1,y1 = ij1
    for i in range(n):
      for j in range(n):
          echange2pix(image, (x0+i, y0+j),
          (x1+i, y1+j))

def tourneRecursifQuadrant(image,ij0,taille):
    if taille < 2: return
    x,y = ij0
    n = taille // 2
    tourneRecursifQuadrant(image,(x,y),n)
    tourneRecursifQuadrant(image,(x+n,y),n)
    tourneRecursifQuadrant(image,(x+n,y+n),n)
    tourneRecursifQuadrant(image,(x,y+n),n)
    echangeQuadrant(image,(x,y),(x+n,y),n)
    echangeQuadrant(image,(x+n,y),(x+n,y+n),n)
    echangeQuadrant(image,(x+n,y+n),(x,y+n),n)

def tourneRecursif(image):
    assert(image.size[0] == image.size[1])
    tourneRecursifQuadrant(image,(0,0),image.size[0])
```

- La fonction `echangeQuadrant` échange les quadrants dont les coordonnées sont `ij0` et `ij1`.
- `tourneRecursifQuadrant` tourne chaque quadrant de `image` d'un quart de tour, puis translate chaque quadrant en permutant circulairement chaque pixel de $\frac{n}{2}$ comme sur la figure ci-contre.



rotation récursive des quadrants — permutation circulaire des quadrants

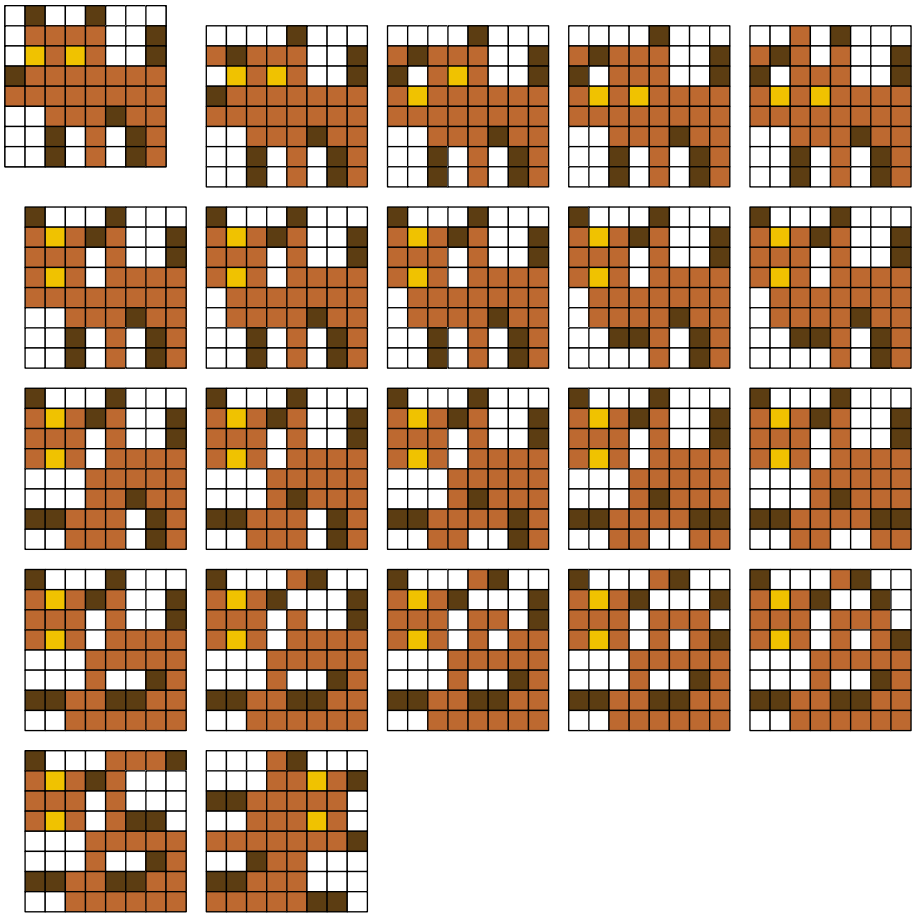


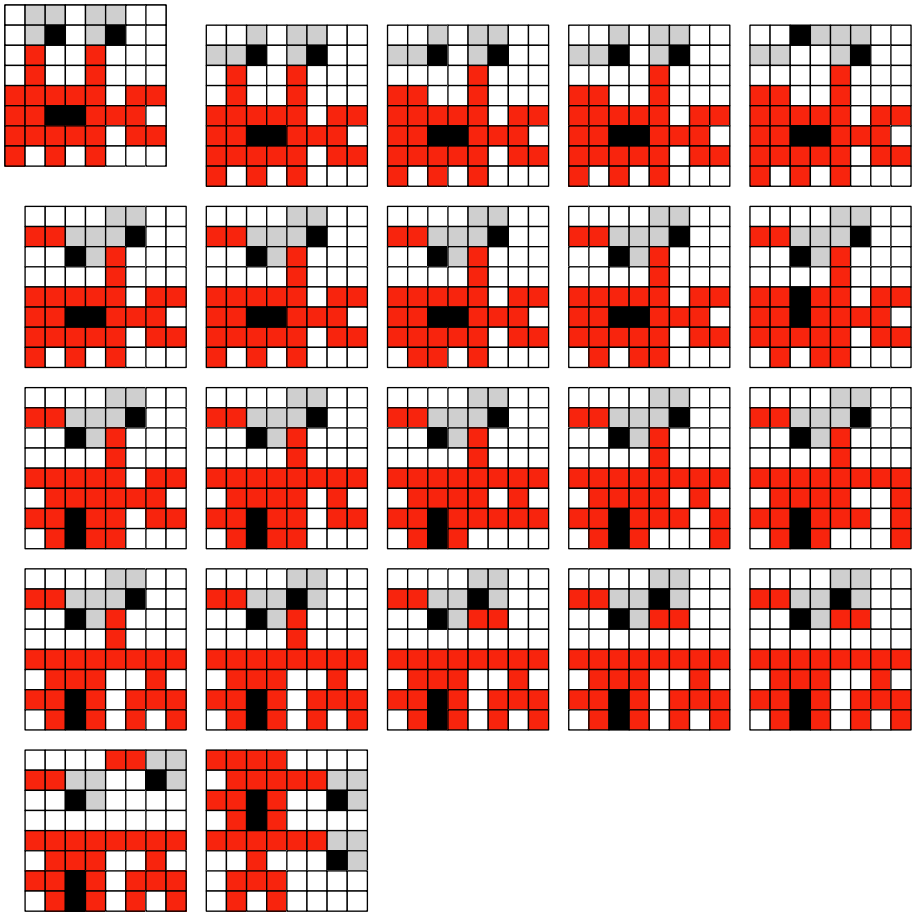
Complexité de l'algorithme

La compléxité vérifie $\mathcal{C}(n) = 4\mathcal{C}(n/2) + \mathcal{O}(1) = \mathcal{O}(n^2)$ ce qui mène, si $n = 2^p$ à :
$\mathcal{C}(2^p) = 2^{2p} + 4 \times 2^{2p-2} + C^2 \times 2^{2p-4} + \cdots + 4^{p-1}2^2 = p \times 4^p = p \times 2^{2p} = \mathcal{O}(\log_2(n)n^2)$
Avec $\mathcal{O}(1)$ mémoire auxiliaire puisque c'est un traitement en place (sans recopie).

Travail à faire :

Faire tourner l'algorithme sur la même image que précédemment et notez le temps de traitement : .....
Que peut-on en conclure ?
.....
.....





Voir également la vidéo : https://www.youtube.com/watch?v=0Xo-uzzD4Js