TP - Diviser pour régner

M. Tellene

Organisation : dans votre dossier personnel, créer un dossier « diviser pour regner », c'est ce dossier qui contiendra le travail fait lors de ce TP

1 Quelques rappels

La méthode de programmation « Diviser pour régner » suit le principe suivant :

1. décomposer un problème à résoudre en sous-problèmes, plus petits, puis à les résoudre, éventuellement en appliquant le même principe autant de fois que nécessaire
2. combiner les résultats des sous-problèmes pour en déduire le résultat du problème initial

2 Une aide pour tester

Afin de tester vos fonctions, pour vous générer des tableaux aléatoires de la manière suivante :

```
from random import randint

tab = [randint(0, 10) for _ in range(5)]
#ici on génére un tableau de 5 éléments
#les valeurs sont comprises entre 0 et 10
```

3 A votre tour

Exercice 1
Écrire une version récursive de la recherche dichotomique

Pour rappel, la recherche dichotomique se déroule comme suit :

1. on compare la valeur `x` à chercher dans `tab` avec le milieu de l'intervalle de recherche (au début l'intervalle est *[0..taille -1]*)
2. si la valeur du milieu est plus grande que `x`, alors on décale l'intervalle vers la gauche
3. sinon si la valeur du milieu est plus petite que `x`, on décale l'intervalle vers la droite
4. on recommence tant qu'on n'a pas trouvé la valeur **et** que l'intervalle de recherche contient au moins une valeur

Exercice 2
Écrire une version récursive du tri fusion

Pour rappel, le tri fusion se déroule comme suit :

1. on coupe le tableau en deux récursivement jusqu'à avoir des sous-tableaux d'un élément

2. on fusionne les sous-tableaux en un, en triant les éléments

Une petite aide :

1. Commencer pour écrire une fonction `coupe(tab)` qui sépare un tableau en deux sous-tableaux de même taille (plus ou moins un élément)
2. Ensuite, écrire une fonction récursive `fusion(t1, t2)` qui fusionne `t1` et `t2` dans un tableau trié
3. Enfin, écrire une fonction `tri_fusion(tab)` qui trie le tableau en suivant la méthode fusion (garder bien à l'esprit que l'on cherche à décomposer le tableau en sous-tableaux d'un seul élément)

Exercice 3

Cet exercice va s'intéresser à la comparaison du temps d'exécution du tri sélection et tri fusion.

1. Écrire une fonction `tri_selection` qui tri un tableau suivant la méthode sélection
2. Nous allons mesurer le temps d'exécution des deux algorithmes
   (a) Importer `time` du module `time` de la manière suivante : `from time import time`
   (b) Recopier **et comprendre** le code suivant :

```
def mesure_temps(n):
    tab = [randint(0,10) for _ in range(n)]
    tab2 = tab.copy()

    debut = time()
    tri_fusion(tab)
    fin = time()
    duree_fusion = fin - debut
    print("Temps d'exécution de fusion :", duree_fusion)

    debut = time()
    tri_selection(tab)
    fin = time()
    duree_selection = fin - debut
    print("Temps d'exécution de sélection :", duree_selection)
```

   (c) Comparer les temps d'exécutions des deux méthodes de tri en faisant varier n
3. Afin d'avoir un rendu plus propre, nous allons tracer des courbes pour mettre en évidence la différence de complexité
   (a) Vérifier que le module `mathplotlib` est installé. Pour se faire, essayer d'importer le module de la manière suivante :

```
import mathplotlib.pyplot as plt
```

   Si cela provoque une erreur, alors il va falloir installer le module

   Ouvrir le terminal et taper `pip install matplotlib` ou `pip3 install matplotlib` (cela dépend des versions)
   (b) Une fois le module installé, écrire une fonction `test()` qui :
   — initialise un tableau `durees` avec les valeurs suivantes : 100, 500, 1000, 2000, 4000, 8000 et 16000

— crée deux tableaux vides : `durees_fusion` et `durees_selection`
— parcourt `durees`, lance une mesure de temps pour $n = duree[i]$, récupère les temps d'exécutions calculés et met celui du tri fusion dans `durees_fusion` et celui du tri sélection dans `durees_selection`

Pour que tout cela fonctionne, il va falloir modifier la fonction `mesure_temps(n)` pour qu'elle n'affiche plus les temps mais qu'elle les renvoie

(c) **Sous la fonction `test()`**, recopier le code suivant :

```
def line_plot(fusion, selection):
    x = [100, 500, 1000, 2000, 4000, 8000, 16000]
    courbe1 = fusion
    courbe2 = selection

    _, ax = plt.subplots()
    ax.plot(x, courbe1, label="tri fusion")
    ax.plot(x, courbe2, label="tri sélection")
    plt.title("Tri fusion vs Tri sélection")
    ax.legend()

    plt.show()

if __name__ == "__main__":
    durees_fusion, durees_selection = test()
    line_plot(durees_fusion, durees_selection)
```

(d) Lancer votre programme et attendre le résultat (cela peut prendre un moment)

Exercice 4

Dans cet exercice, on cherche à effectuer la rotation d'image de 90 degrés. Cette manipulation peut se faire via la méthode « Diviser pour régner ». En effet, il est plus simple de faire une rotation d'une sous-partie de l'image puis de remettre les morceaux aux bons endroits que de faire une rotation de l'image d'un coup

Pour manipuler des images avec Python, nous utiliserons le module `Image` de la bibliothèque `PIL`

1. Recopier le code suivant :

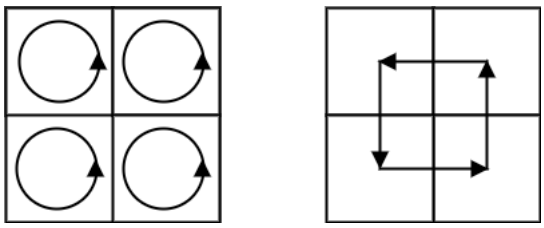
```
from PIL import Image

im = Image.open("image.png")
largeur, hauteur = im.size
px = im.load()
```

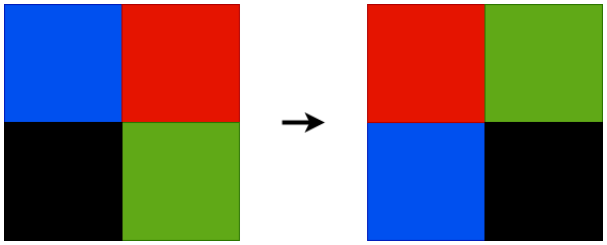
Ce code permet de charger l'image contenue dans le fichier *image.png*, on obtient ses dimensions dans les variables `largeur` et `hauteur`, et la variable `px` est la matrice des pixels constituant l'image

Pour $0 \leq$ `x` $\leq$ `largeur` et $0 \leq$ `y` $\leq$ `hauteur`, la couleur du pixel (x, y) est donnée par `px[x,y]`. Une couleur est un triplet suivant l'ordre $(rouge, vert, bleu)$ sous forme d'entier de 0 à 255

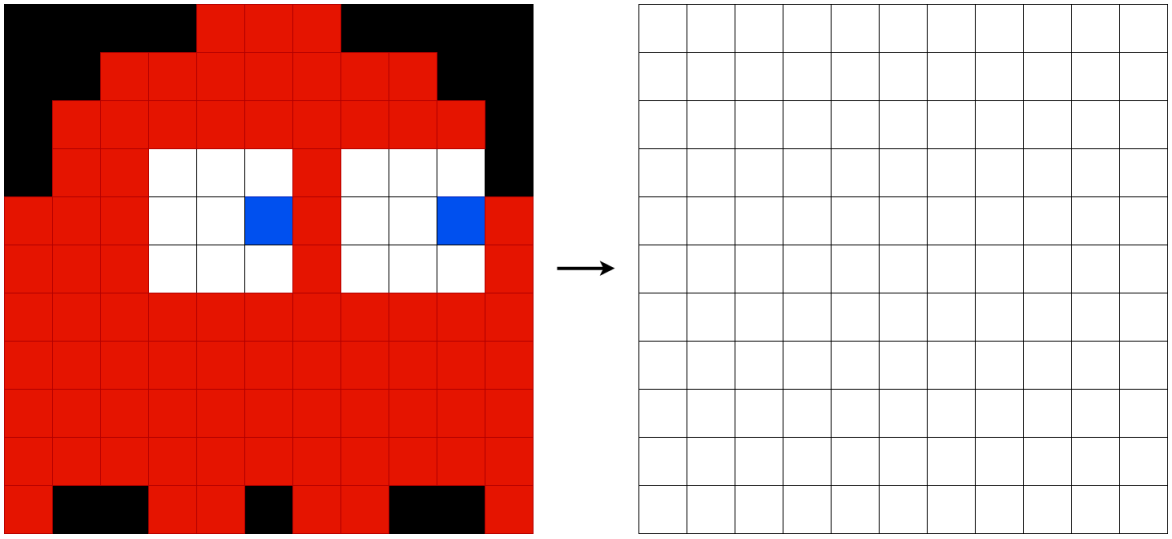
Dans cet exercice on suppose que **l'image est carrée et que sa dimension est une puissance de deux**. Notre idée consiste à découper en quatre, à effectuer une rotation de 90° de chacun des quatre morceaux, puis à les déplacer vers leur position finale :



Avec un exemple concret, cela donne :



2. Faire la rotation à 90 degré de l'image ci-dessous :



3. Afin de pouvoir procéder récursivement, écrire une fonction `rotation_aux(px, x, y, t)` qui effectue une rotation de la portion carrée de l'image comprise entre les pixels (x, y) et $(x + t, y + t)$. Cette fonction ne doit rien renvoyer

   Cette fonction modifie le tableau `px` pour effectuer la rotation de cette portion de l'image au même endroit. On suppose que `t` est une puissance de 2

   (a) penser au cas d'une région à un pixel unique

   (b) puis découper en 4 régions avec un appel récursif

   (c) enfin, déplacer les 4 régions :

Le code à trou suivant reprend les 3 étapes précédentes, à vous de la compléter

```
def rotation_aux(px, x, y, t):
    if t==1:
        ..........................................................................
    t = ..........................................................................
    rotation_aux(px, ......................., ......................., t)
    rotation_aux(px, ......................., ......................., t)
    rotation_aux(px, ......................., ......................., t)
    rotation_aux(px, ......................., ......................., t)

    for i in range(x, x+t):
        for j in range(y, y+t):
            px[..., ...], px[..., ...], px[..., ...], px[..., ...] = \
            px[...., ....], px[...., ....], px[...., ....], px[...., ....]
```

4. Écrire une fonction `rotation(px, taille)` qui effectue une rotation de l'image toute entière

Comment voir votre résultat ? Utiliser l'instruction `im.show()` [1]

1. `im` contient l'image chargée (cf. premier code donné dans l'exercice)