Diviser pour régner

19 mars 2026

Introduction

Programme

Méthode « diviser pour régner » : **Écrire un algorithme utilisant la méthode « diviser pour régner ».**

La rotation d'une image bitmap d'un quart de tour avec un coût en mémoire constant est un bon exemple.

L'exemple du tri fusion permet également d'exploiter la récursivité et d'exhiber un algorithme de coût en $n \log_2 n$ dans les pires des cas.

Exemple

On veut trier un jeu de cartes avec les règles suivantes :

- coeur $>$ carreau $>$ trefle $>$ pique
- as $>$ roi $>$ dame $>$ valet $>$ 10 $>$ 9 $>$ 8 ... $>$ 2

Méthode :

- on divise le paquet avec autant de paquets que de personne
- chaque personne trie son paquet
- on **fusionne** les paquets triés (comment ?)

Chaque personne peut éventuellement “sous-traiter” le tri de la même manière.

Diviser pour régner

Le principe des algorithmes de type **diviser pour régner** consiste à

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

Les techniques de diviser pour régner sont souvent utilisée de manière récursives.

Un autre avantage des algorithmes de type “diviser pour régner” est qu'ils peuvent rendre parallèles des problèmes et ainsi optimiser l'utilisation du processeur.

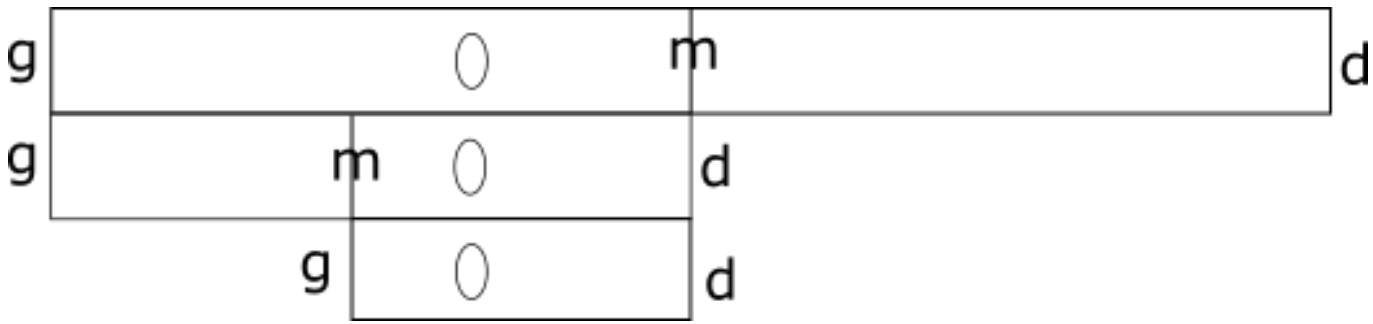
Recherche dichotomique

Principe

L'objectif est de déterminer si une valeur est présente dans un tableau trié (exemple : la liste de tous les mots du français).

On peut décomposer le problème en un sous-problème : est-ce que la valeur `v` recherchée est dans le sous-tableau commençant à l'indice `g` et finissant à l'indice `d`.

Algorithme I



Algorithme II

On regarde alors la valeur du centre de cet intervalle (indice $m = g + \frac{d-g}{2} = \frac{g+d}{2}$

- si cette valeur est égale à `v`, on retourne l'information qu'on a trouvé la valeur
- si cette valeur est supérieure, on sait qu'il faut chercher dans le sous-tableau gauche (entre `g` et `m`)
- si cette valeur est inférieure à `v`, on sait qu'il faut chercher dans le sous-tableau droit (entre `m` et `d`)

Implémentation

```
from typing import List, Union
def dicho(v: str, t: List[str],
          g:int , d: int) -> Union[int, None]:
    if g > d:
        return None
    m = (g+d)//2
    if t[m] < v:
        return dicho(v, t, m+1, d)
    if t[m] > v:
        return dicho(v, t, g, m-1)
    return m

def recherche(v, t):
    return dicho(v, t, 0, len(t)-1)
```

Efficacité

La recherche purement séquentielle dans un tableau a un coût de N.

La recherche dichotomique divise l'intervalle par deux à chaque itération ; on a donc au pire $2^i = N$ soit $i = \ln N$ itérations

On ne risque pas de `RecursionError` : pour une profondeur de 1000 récursions, on aurait un tableau initial de $2^{1000} = 10^{301}$ éléments, largement plus grand que la mémoire de n'importe quel ordi.

Exercices I

En vous inspirant des instructions suivantes

```
import random
N = 1000
liste = sorted([random.randint(0, 2*N) for _ in range(N)])

import timeit
start = timeit.default_timer()
# code à chronométrer
duree = timeit.default_timer() - start
print(f"Durée : {duree:3.3}s")
```

Exercices II

```
import matplotlib.pyplot as plt
%matplotlib inline
Ns = [1,2,3,4,5]
Ts = [2,4,6,8,10]
Tr = [1,1,1,1,1]

plt.figure()
plt.plot(Ns,Ts,color='b', marker = '+')
plt.plot(Ns,Tr,color='r', marker = '+')

plt.xlabel("N")
plt.ylabel("Temps (s)")
plt.grid()
plt.title("Evolution du temps en fonction de N")
plt
```

Exercices III

Comparer numériquement puis graphiquement le temps de recherche moyen d d'un élément $\mathtt{v}$ aléatoire, pour différentes tailles N de listes, entre une simple recherche séquentielle et la recherche dichotomique proposée au dessus.

Evaluez le coût de chacune en calculant d'une part $\frac{temps}{N}$ et $\frac{temps}{\ln N}$

Tri fusion

Algorithme

cf introduction

Implémentation tri I

```
from typing import List, Tuple
def tri(l: List[int],
        g:Union[int, None]=None,
        d:Union[int, None]=None,
        r:Union[List[int], None]=None) \
        -> Tuple[int,int]:
    """

    :param l:
    :param g:
    :param d:
    :param r:
    :return:
    """
```

Implémentation tri II

```
# Appel initial
if g is None:
    g=0
if d is None:
    d = len(l)-1
if r is None:
    r = l[:]
```

Implémentation tri III

```
# Terminaison récusive
if d == g:
    return g, d
if d == g + 1:
    if l[g] > l[d]:
        l[d], l[g] = l[g], l[d]
    return g, d
```

Implémentation tri IV

```
# Cas récursif
m = (g+d)//2
tri(l, g, m, r)
tri(l, m+1, d, r)
return fusion(l, g, m, m+1, d, r)
```

Implémentation fusion I

```
from typing import List, Tuple
def fusion(l: List[int],
           g1: int,
           d1: int,
           g2: int,
           d2: int,
           r: List[int]) -> Tuple[int, int]:
    """

    :param l:
    :param g1:
    :param d1:
    :param g2:
    :param d2:
```

Implémentation fusion II

```
    :param r:
    :return:
    """
    p1 = g1
    p2 = g2
    p = g1
    while p <= d2:
        # Cas où une la fin d'une des
        # deux listes est atteinte
        if p2 > d2:
            r[p] = l[p1]
            p1 += 1
        elif p1 > d1:
            r[p] = l[p2]
            p2 += 1
```

Implémentation fusion III

```
        # Sélection de l'élément le
        # plus petit
        elif l[p1] < l[p2]:
            r[p] = l[p1]
            p1+=1
        elif l[p1] >= l[p2]:
            r[p] = l[p2]
            p2+=1
        p+=1
    # Recopie des éléments triés dans
    # la liste l
    for i in range(d2-g1+1):
        l[g1+i] = r[g1+i]
    return g1, d2
```

Exercice 1 : compréhension

Examinez (print / pythontutor..) le programme ci dessus agissant sur la liste `[34, 20, 11, 42, 98, 13, 52, 18, 11]`

Pour afficher une liste sur une seule ligne, vous pouvez utiliser `l.join(" ")`

Complétez les docstrings des deux fonctions avec les explications sur le fonctionnement, les paramètres et le return.

Exercice 2 : efficacité I

Comparez numériquement puis graphiquement pour différentes valeurs de N le coût du tri fusion par rapport à d'autres algorithmes de tri : tri à bulles, tri par insertion, tri par sélection.

Pour référence, les programmes de tri par insertion / sélection / bulle sont donnés. Profitez en pour jeter un coup d'oeil sur les caractéristiques de chacun de ces algorithmes de tri (programme de 1ère), et de manière plus générale jetez un coup d'oeil sur
https ://visualgo.net/en/sorting ?slide=1

Exercice 2 : efficacité II

```
def tri_insertion(tab):
    # Parcour de 1 à la taille du tab
    for i in range(1, len(tab)):
        k = tab[i]
        j = i-1
        while j >= 0 and k < tab[j] :
                tab[j + 1] = tab[j]
                j -= 1
        tab[j + 1] = k
```

Exercice 2 : efficacité III

```
def tri_selection(tab):
    for i in range(len(tab)):
        # Trouver le min
        min = i
        for j in range(i+1, len(tab)):
            if tab[min] > tab[j]:
                min = j

        tmp = tab[i]
        tab[i] = tab[min]
        tab[min] = tmp
    return tab
```

Exercice 2 : efficacité IV

```
def tri_bulle(tab):
    n = len(tab)
    # Traverser tous les éléments du tableau
    for i in range(n):
        for j in range(0, n-i-1):
            # échanger si l'élément trouvé
            # est plus grand que le suivant
            if tab[j] > tab[j+1] :
                tab[j], tab[j+1] = tab[j+1], tab[j]
```

Rotation d'une image

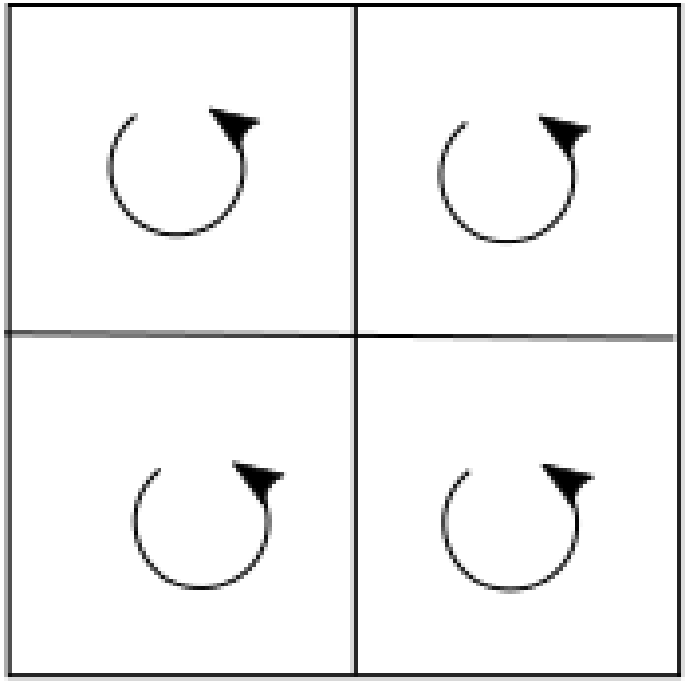
Principe I

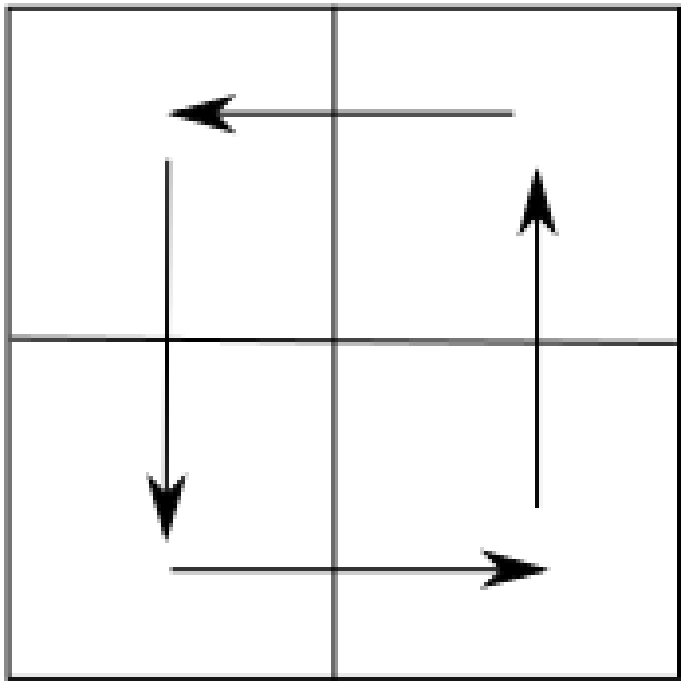
On peut utiliser le principe “diviser pour régner” pour effectuer la rotation de 90 degrés d'une image à espace mémoire constant.

On se placera dans le cas d'une image carrée dont les dimensions sont une puissance de deux (256x256 par exemple).

Le principe est de couper l'image en 4, à effectuer la rotation de 90 degrés de chacun des 4 morceaux, puis de les déplacer vers leurs positions finales.

Principe II





Implémentation I

On charge une image en utilisant la librairie PIL (Python Image Library)

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

Chaque pixel (R,V,B) ou (R,V,B,A) est accessible comme un triplet ou quadruplet dans `p[x, y]` avec `0 <= x < largeur` et `0 <= y < hauteur`. On peut modifier un pixel en écrivant dans ce tableau : `p[x,y] = c` avec `c` un triplet RGB ou quadruplet RGBA.

Implémentation II

Ecrire une fonction récursive `rotation(px, t, x, y)` qui effectue la roation de la portion carrée de l'image comprise entre les pixels (x,y) et (x+t, y+t)

Indication pour le déplacement : en python, on peut effectuer une assignation multiple : `a,b,c,d = d,a,c,b`.