Element majoritaire

vendredi 3 mars 2017 14:24

Question 1

```
def element_majoritaire(tableau):
    dictionnaire = {}
    for element in tableau:
        if element in dictionnaire:
            dictionnaire[element] += 1
        else:
            dictionnaire[element] = 1
        if dictionnaire[element] > len(tableau)/2:
            return element
```

Question 2

```
def occurences(talbeau, a_compter):
    nombre = 0
    for element in range(talbleau):
        if element == a_compter:
            nombre += 1
    return nombre

def element_majoritaire_2(talbeau):
    for element in tableau:
        if occurences(tableau, element) > len(tableau)/2:
            return element
```

Cet algorithme est de complexité n^2.

Optimisations :
- On n'a besoin que de parcourir que la moitié du tableau ! En effet, si élément majoritaire il y a, il apparaît nécessairement dans la première moitié
- On peut mélanger le tableau avant de le parcourir. Exemple : 0001111. Il faut parcourir les trois 0 avant d'arriver à l'élément majoritaire, ce qui est peu efficace. Par contre, si on mélange, on a une chance sur 2 de tomber tout de suite sur l'élément majoritaire, ce qui met fin à l'algorithme.

Question 3

```
def element_majoritaire_dpr(tableau, debut, fin):
    if debut == fin:
        return tableau[debut]
    majoritaire_debut = element_majoritaire_dpr(tableau, debut, (debut + fin)//2)
    majoritaire_fin = element_majoritaire_dpr(tableau, (debut + fin)//2 + 1, fin)
    #On n'a plus que 2 candidats pour l'element majoritaire, plus qu'a les tester
    for candidat in majoritaire_debut, majoritaire_fin:
        nombre = 0
        for element in tableau:
```

```
            if element == candidat:
                nombre += 1
        if nombre > len(tableau)/2:
            return candidat
```

Question 4

Complexité en $n \log n$.
En effet, en notant $C(n)$ le coût pour un tableau de taille n, on a :
$C(n) = 2C\left(\frac{n}{2}\right) + O(n)$
On conclue en appliquant le master theorem.

Question 5

$\underbrace{10}_{x}\ \underbrace{11}_{1}\ \underbrace{11}_{1}\ \underbrace{21}_{x}$
$\Rightarrow$ on réitère sur 11

Question 6

On obtient un mauvais élément.

Question 7

Si le tableau est de taille impaire, on utilise le dernier élément pour "départager", c'est-à-dire qu'on le garde à chaque fois. Mais attention, il faut vérifier à la fin par un parcours du tableau que le résultat est bien majoritaire.

Question 8

$A = A_1 A_2, B = B_1 B_2$ de longueur n
$A \times B = A_1 \times B_2 \times 2^n + A_2 \times B_2$
$+A_1 \times B_2 \times 2^{\frac{n}{2}} + A_2 \times B_1 \times 2^{\frac{n}{2}}$

```
def multiplier(A, B):
    n = len(A)
    if n == 1:
        return A[0]*B[0]
    A1, A2 = diviser(A)
    B1, B2 = diviser(B)
    A1B1 = multiplier(A1, B1)
    A2B2 = multiplier(A2, B2)
    A1B2 = multiplier(A1, B2)
```

```
    A2B1 = multiplier(A2, B1)
    return decalage(A1B1, n) + A2B2 +
decalage(A1B2, n/2) + decalage(A2B1, n/2)
```

Question 10

La complexité est en $n^{\log 4} = n^2$, ce qui est autant efficace que l'algorithme classique !

Question 11

Les 3 multiplications à faire sont :
$M_1 = A_1 B_1$
$M_2 = A_2 B_2$
$M_3 = (A_1 + A_2)(B_1 + B_2)$

On a ainsi :
$A \times B = M_1 \times 2^n + M_2 + 2^{\frac{n}{2}}(M_3 - M_1 - M_2)$

La complexité est en $O\left(n^{\log_2 3}\right)$
C'est plus efficace que l'algorithme classique !