TD d'algorithmique avancée

TD3 : L'élément majoritaire

Soit un tableau ***A*** de ***n*** éléments, tel que $n=2^k$. Un élément de ***A*** est majoritaire s'il a une fréquence supérieure à (***n***/2).

I. Nous cherchons le nombre d'occurrences d'un élément dans un tableau d'une manière simple.

1. Proposez un algorithme simple sous forme d'une fonction appelée « OCCURRENCES » qui calcule le nombre d'occurrences d'une valeur ***a*** entre les cases ***i*** et ***j*** du tableau ***A***.
2. Calculez ça complexité en nombre d'itérations.
3. A l'aide de la fonction « OCCURRENCES », proposez un algorithme sous forme d'une fonction appelée « MAJORITAIRE » qui vérifie si le tableau ***A*** contient un élément majoritaire.
4. Calculez ça complexité en nombre d'itérations.

II. Nous cherchons à optimiser notre algorithme, pour cela nous allons procéder comme suit :

1. Proposez un algorithme pour la fonction « MAJORITAIRE » qui reçois un tableau ***A*** et deux indices ***d, f*** et retourne le couple (***Vrai, x***) si le tableau ***A*** contient l'élément majoritaire ***x*** et renverra le couple (***Faux, 0***) si le tableau ***A*** ne contient pas d'élément majoritaire. Cette fonction divisera en deux le tableau ***A*** et cherche le majoritaire de chaque moitié à part en utilisant la fonction `OCCURRENCES()`.
2. Calculez la complexité de cet algorithme en nombre d'appels à la fonction `OCCURRENCES()`.

TD d'algorithmique avancée

Corrigé du TD2 : La récursivité

Soit un tableau ***A*** de ***n*** éléments, tel que $n=2^k$. Un élément de ***A*** est majoritaire s'il a une fréquence supérieure à (***n***/2).

I. Nous cherchons le nombre d'occurrences d'un élément dans un tableau d'une manière simple.

1. Proposez un algorithme simple sous forme d'une fonction appelée « OCCURRENCES » qui calcule le nombre d'occurrences d'une valeur ***a*** entre les cases ***i*** et ***j*** du tableau ***A***.

```
OCCURRENCES(x, A, i, j)
   compteur ← 0
   pour k ← i à j faire
         si A[k] = x alors compteur ← compteur + 1
   retourner compteur
```

2. Calculez ça complexité en nombre d'itérations.

La complexité : O(j-i) = j-i+1

3. A l'aide de la fonction « OCCURRENCES », proposez un algorithme sous forme d'une fonction appelée « MAJORITAIRE » qui vérifie si le tableau ***A*** contient un élément majoritaire.

```
MAJORITAIRE(A)
   pour i ← 1 à taille(A)/2 faire
      si OCCURRENCES(A[i], A, i, taille(A)) > taille(A)/2 alors
            retourner True
   retourner False
```

4. Calculez ça complexité en nombre d'itérations.

La complexité :

1ère itération: n-1+1 = n

2ème itération: n-2+1 = n-1

3ème itération: n-3+1 = n-2

(n/2)ème itération: n-n/2+1 = n/2 + 1

La somme d'une suite arithmétique $= \frac{\text{nombre d'éléments}}{2} \cdot \left(1^{er}\text{ élément} + \text{dernier élément}\right)$

$$= \frac{n/2}{2} \cdot \left(n + n/2 + 1\right) = \frac{1}{8} \cdot \left(3 \cdot n^2 + 2 \cdot n\right)$$

Donc $O\left(n^2\right) = \frac{1}{8} \cdot \left(3 \cdot n^2 + 2 \cdot n\right)$

$O(n^2) = (1/8)(3n^2 + 2n)$

II. Nous cherchons à optimiser notre algorithme, pour cela nous allons procéder comme suit :

1. Proposez un algorithme pour la fonction « MAJORITAIRE » qui reçois un tableau ***A*** et deux indices ***d***, ***f*** et retourne le couple (***Vrai, x***) si le tableau ***A*** contient l'élément majoritaire ***x*** et renverra le couple (***Faux, 0***) si le tableau ***A*** ne contient pas d'élément majoritaire. Cette fonction divisera en deux le tableau ***A*** et cherche le majoritaire de chaque moitié à part en utilisant la fonction `OCCURRENCES()`.

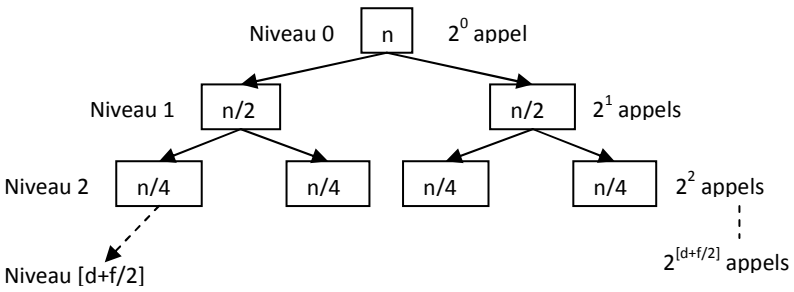
```
MAJORITAIRE(A, d, f)
      si d=f alors retourner(Vrai, A[d])
      m ← [(d+f)/2]
      (bx, x) ← MAJORITAIRE(A, d, m)
      (by, y) ← MAJORITAIRE(A, m+1, f)
      si bx=Faux et by=Faux alors retourner(Faux, 0)
      si bx=Vrai et by=Vrai alors
            si x=y alors retourner(Vrai, x)
            sinon  cx ← OCCURRENCES(x, A, d, f)
                   cy ← OCCURRENCES(y, A, d, f)
                   si cx > (f-d+1)/2 alors retourner(Vrai, x)
                   sinon  si cy > (f-d+1)/2 alors retourner(Vrai, y)
                          sinon retourner(Faux, 0)
      sinon  si bx=Vrai alors
                   si OCCURRENCES(x, A, d, f) > (f-d+1)/2 alors retourner(Vrai, x)
                   sinon retourner(Faux, 0)
            sinon  si OCCURRENCES(y, A, d, f) > (f-d+1)/2 alors retourner(Vrai, y)
                   sinon retourner(Faux, 0)
```

La fonction **`OCCURRENCES(x,A,d,f)`** est celle de la première question.

2. Calculez la complexité de cet algorithme en nombre d'appels à la fonction `OCCURRENCES()`.

L'algorithme fait au maximum 2 appels à la fonction `OCCURRENCES()` qui contient les itérations des boucles.

Les appels à la fonction `MAJORITAIRE()` se font sous forme d'un arbre binaire comme suit :



tel que 'n' est la taille du tableau A

Nombre total d'appels à la fonction `MAJORITAIRE()` au maximum = $2^0+2^1+2^2+\ldots+2^{[d+f/2]}$

= somme d'une suite géométrique = $u_0 \cdot \frac{1-b^{\text{nombre d'éléments}}}{1-b} = 1 \cdot \frac{1-2^{\left[\frac{d+f}{2}\right]+1}}{1-2} = 2^{\left[\frac{d+f}{2}\right]+1} - 1$

La complexité donc est de l'ordre de : $O(2^{(d+f)/2}) = 2.(2^{[(d+f)/2]+1}-1)$ (Complexité exponentielle)