# Algorithmique Avancée

Denis TRYSTRAM

Elément majoritaire

sept. 2022

# Définition

- Un tableau possède un élément majoritaire si plus de la moitié (strictement) de ses entrées sont identiques.

Etant donné un tableau $A$ à $n$ éléments, on se propose de **concevoir un algorithme pour déterminer s'il possède un élément majoritaire**, et dans l'affirmative de **l'identifier**.

Les éléments du tableau ne sont pas forcément comparables et il n'est pas possible d'effectuer des comparaisons de la forme $A[i] > A[j]$ (on peut penser par exemple à des images).
Par contre, on peut répondre à des questions de la forme $A[i] = A[j]$? en temps constant.

# Définition

- Un tableau possède un élément majoritaire si plus de la moitié (strictement) de ses entrées sont identiques.

Etant donné un tableau $A$ à $n$ éléments, on se propose de **concevoir un algorithme pour déterminer s'il possède un élément majoritaire**, et dans l'affirmative de **l'identifier**.

Les éléments du tableau ne sont pas forcément comparables et il n'est pas possible d'effectuer des comparaisons de la forme $A[i] > A[j]$ (on peut penser par exemple à des images).
Par contre, on peut répondre à des questions de la forme $A[i] = A[j]$? en temps constant.

On peut commencer à remarquer que s'il existe, un élément majoritaire est nécessairement unique...

# Questions

- Ecrire un algorithme qui calcule le nombre d'occurrences d'un élément donné dans un tableau.
- En déduire un algorithme simple pour trouver l'élément majoritaire.
- Evaluer son coût.

- La première idée qui vient est de considérer les éléments les uns après les autres et de vérifier leurs occurrences dans le tableau.

- Le coût au pire cas de cet algorithme est dans $\mathcal{O}(n^2)$ pour un tableau de $n$ éléments.

Remarque : on peut améliorer marginalement l'algorithme en marquant les éléments déjà visités, comme dans un crible (mais le pire cas reste du même ordre).

# Diviser pour régner

Partitionner le tableau $A$ en deux tableaux $A1$ et $A2$ de même taille
On suppose que la taille du tableau initial est une puissance de 2.

S'il existe un élément majoritaire dans $A$ alors il est majoritaire dans au moins un des deux sous-tableaux.
On en déduit un algorithme récursif qui calcule les éléments majoritaires dans chacun des deux sous-tableaux (s'ils existent) et leurs nombres d'occurrences.

- Détailler cet algorithme.

# Fonction MAJ

On propose d'écrire une procédure MAJ(i,j) qui renvoie un couple $(x, c_x)$ si $x$ est majoritaire dans le sous-tableau $A(i..j)$ avec un nombre d'occurrences $c_x$
il renvoie $(-, 0)$ s'il n'y a pas d'élément majoritaire.

# Fonction MAJ

On propose d'écrire une procédure MAJ(i,j) qui renvoie un couple $(x, c_x)$ si $x$ est majoritaire dans le sous-tableau $A(i..j)$ avec un nombre d'occurrences $c_x$
il renvoie $(-, 0)$ s'il n'y a pas d'élément majoritaire.

Principe de la phase de fusion :

- aucun des deux sous-tableaux ne possède d'élément majoritaire et dans ce cas, il n'y en a pas
- un seul des deux possède un élément majoritaire et alors, il suffit de parcourir l'autre sous-tableau pour compter ses occurrences (coût n/2)
- A1 et A2 ont chacun un élément majoritaire, il faut alors tester les occurrences sur chaque sous-tableau

# Algorithme (tableau $E$)

**Algorithme 2** $Majoritaire(i, j)$

**si** $i = j$ **alors**
  **retourner** $(E[i], 1)$
**sinon**
  $(x, c_x) = Majoritaire(i, \lfloor (i+j)/2 \rfloor)$
  $(y, c_y) = Majoritaire(\lfloor (i+j)/2 \rfloor + 1, j)$
  **si** $c_x \neq 0$ **alors**
    $c_x \leftarrow c_x + Occurence(x, \lfloor (i+j)/2 \rfloor + 1, j)$
  **si** $c_y \neq 0$ **alors**
    $c_y \leftarrow c_y + Occurence(x, i, \lfloor (i+j)/2 \rfloor)$
  **si** $c_x > \lfloor (j-i+1)/2 \rfloor$ **alors**
    retourner $(x, c_x)$
  **sinon**
    **si** $c_y > \lfloor (j-i+1)/2 \rfloor$ **alors**
      **retourner** $(y, c_y)$
    **sinon**
      **retourner** $(-, 0)$

- Conclure en donnant un algorithme complet et calculer son coût en pire cas.

# Quel est son coût ?

- On donnera l'équation de récurrence et on explicitera comment la résoudre.

# Quel est son coût ?

- On donnera l'équation de récurrence et on explicitera comment la résoudre.

L'expression du coût (exprimé en nombre de comparaisons) au pire cas est :

- $T(1) = 1$
- $T(n) = 2(T(n/2) + \frac{n}{2})$

Le calcul est exactement celui du tri fusion et de l'enveloppe convexe : $n.log_2(n)$

- Retrouver ce résultat par application du master theorem et par le recursive tree directement.

# Amélioration

On cherche à construire un algorithme avec la propriété suivante :

## Propriété

- Soit l'algorithme garantit que le tableau $A$ n'a pas d'élément majoritaire
- Soit il fournit un indice $p > \frac{n}{2}$ et un élément $x$ tel que l'occurence de $x$ soit au plus $p$ et tel que tout autre élément apparaisse au plus $n - p$ fois

# Amélioration

On cherche à construire un algorithme avec la propriété suivante :

## Propriété

- Soit l'algorithme garantit que le tableau $A$ n'a pas d'élément majoritaire
- Soit il fournit un indice $p > \frac{n}{2}$ et un élément $x$ tel que l'occurence de $x$ soit au plus $p$ et tel que tout autre élément apparaisse au plus $n - p$ fois

- Montrer qu'un tel $x$ est un candidat majoritaire
- Dérouler l'algorithme sur un exemple $n = 8$

# Algorithme (tableau $E$)

**Algorithme 3** $Candidat - majoritaire(E)$

**si** $E$ n'a qu'un élément $x$ **alors**
  **retourner** $(x, 1)$
**sinon**
  couper $E$ en deux tableaux $E_1$ et $E_2$ de taille $n/2$
  appeler $Candidat - majoritaire(E_1)$ qui renvoie `AUCUN` ou $(x, p)$
  appeler $Candidat - majoritaire(E_2)$ qui renvoie `AUCUN` ou $(y, q)$
  suivant la réponse pour $E_1$ et $E_2$, faire
  **si** `AUCUN` et `AUCUN` **alors**
    **retourner** `AUCUN`
  **si** `AUCUN` et $(y, q)$ **alors**
    **retourner** $(y, q + \frac{n}{4})$
  **si** $(x, p)$ et `AUCUN` **alors**
    **retourner** $(x, p + \frac{n}{4})$
  **si** $(x, p)$ et $(y, q)$ **alors**
    **si** $x \neq y$ **alors**
      **si** $p > q$ **alors**
        **retourner** $(x, p + \frac{n}{2} - q)$
      **si** $p < q$ **alors**
        **retourner** $(y, q + \frac{n}{2} - p)$
      **si** $p = q$ **alors**
        **retourner** `AUCUN` // (sinon ce serait $x$ ou $y$ mais $c_x \leq \frac{n}{2}$ et $c_y \leq \frac{n}{2}$)
    **sinon** // $x = y$
      **retourner** $(x, p + q)$

L'algorithme a deux phases :

- déderminer un candidat
- vérification si l'élément final (s'il existe) est bien majoritaire dans le tableau initial.

Phase 1 : La première étape coûte $n/2$ comparaisons, la seconde au plus $n/4$, etc.. Soit $\Theta(n)$ au final.

- $T(1) = 1$
- $T(n) = 2T(n/2) + 1$

Le coût est linéaire.

Phase 2 (vérification) : linéaire en $n$