Algorithmique : DM1

Younesse Kaddar

Énoncé

I. Préliminaires

1

Méthode 1 :

On remarque que la valeur majoritaire est, si elle existe, la $\lfloor n/2 \rfloor$-ième plus petite valeur.

Algorithme

- On obtient la $\lfloor n/2 \rfloor$-ième plus petite valeur avec l'algorithme vu en TD utilisant la médiane des médianes, en $O(n)$.
- Puis : on vérifie que c'est bien une valeur majoritaire ou non en faisant un parcours du tableau pour compter le nombre d'occurrences de cette valeur majoritaire putative, en temps linéaire.
  - *si le nombre d'occurrences est supérieur ou égal à* $\lfloor n/2 \rfloor + 1$: on a bien affaire à une valeur majoritaire : on la retourne.
  - *sinon* : le candidat n'est pas une valeur majoritaire.

Méthode 2 :

(Fortement inspiré de l'algorithme fourni dans la section 2)

On remarque que le nombre d'occurrences de la valeur majoritaire moins le nombre d'occurrences de toutes les autres valeurs est strictement positif.

On prendra garde à vérifier (en $O(n)$) que la valeur majoritaire potentielle trouvée de la sorte en est bien une.

```python
def valeur_majoritaire(T):
  # on remplit une boîte temporaire d'éléments identiques à la valeur majoritaire tempor
aire et rencontrés d'affilée
  n = len(T)
  val_maj_temp = T[0]
  taille_boite_temporaire = 1

  for i in T:
    if val_maj_temp == i:
      taille_boite_temporaire+=1
    else:
      # on apparie l'élément courant (différent de la valeur majoritaire temporaire) ave
c un élément de la boîte (égal à la valeur majoritaire temporaire), qu'on sort de la boî
te
      taille_boite_temporaire -=1

    if taille_boite_temporaire == 0:
      # la boîte est vide : changement de valeur majoritaire temporaire
      val_maj_temp = i
      taille_boite_temporaire = 1

  # on vérifie si la valeur majoritaire temporaire est bien une valeur majoritaire ou no
n

  # le nombre d'occurrences d'une valeur majoritaire moins le nombre d'occurrences de to
utes les autres valeurs est nécessairement strictement positif
  if taille_boite_temporaire > 0:
    taille_boite_temporaire = 0
    for i in T:
      if i == val_maj_temp:
        taille_boite_temporaire+=1
    if taille_boite_temporaire > n//2:
      return val_maj_temp
    else:
      return None
  else:
    return None
```

2.

On note a, b les deux valeurs possibles, nb_a (resp. nb_b) le nombre d'occurrences de a (resp. b).

$nb_a = nb_b$ est impossible, car sinon n serait pair, donc (par exemple) $nb_a > nb_b$, d'où, comme $nb_a + nb_b = n$,

$$nb_a \geq \lfloor n/2 \rfloor + 1$$

3.

Cet algorithme ne fait que $n - 1$ comparaison d'éléments, car le tableau est parcouru à partir du deuxième élément.

```
def valeur_majoritaire2(T):
  COMPT = [[T[0],1] , [None, 0]]

  # on parcourt le tableau à partir du deuxième élément

  for i in T[1:]:
    if i == COMPT[0][0]:
      COMPT[0][1]+=1
    else:
      if COMPT[1][0] is None:
        COMPT[1][0] = i
      COMPT[1][1]+=1

  # comparaison d'entiers pour choisir la valeur majoritaire

  if COMPT[0][1] > COMPT[1][1]:
    return COMPT[0][0]
  else:
    return COMPT[1][0]
```

4.

Pour $n = 5$:

```
def valeur_majoritaire3(T):
  COMPT = [[T[0],1] , [None, 0]]

  # on parcourt le tableau du deuxième élément à l'avant-dernier élément inclus (pas l
e dernier)

  for i in T[1:-1]:
    if i == COMPT[0][0]:
      COMPT[0][1]+=1
    else:
      if COMPT[1][0] is None:
        COMPT[1][0] = i
      COMPT[1][1]+=1

  # comparaison d'entiers pour choisir la valeur majoritaire

  if COMPT[0][1] == COMPT[1][1]:
    # la valeur majoritaire est nécessairement la dernière valeur : on la retourne
    return T[-1]
  # si il n'y a pas égalité de nb_a et nb_b pour les 4 premiers éléments du tableau : nb
_a >= 3 OU nb_b >= 3 : on a la valeur majoritaire
  elif COMPT[0][1] > COMPT[1][1]:
    return COMPT[0][0]
  else :
    return COMPT[1][0]
```

5.

On généralise l'algorithme proposé pour $n = 5$, avec :

$$\forall n \in \mathbb{N}, u_n \overset{\text{def}}{=} 2n + 1$$

```python
def valeur_majoritaire4(T):
  # on vérifie que T est de taille impaire
  assert len(T)%2 == 1

  COMPT = [[T[0],1] , [None, 0]]

  # on parcourt le tableau du deuxième élément à l'avant-dernier élément inclus (pas l
e dernier)

  for i in T[1:-1]:
    if i == COMPT[0][0]:
      COMPT[0][1]+=1
    else:
      if COMPT[1][0] is None:
        COMPT[1][0] = i
      COMPT[1][1]+=1

  # comparaison d'entiers pour choisir la valeur majoritaire

  if COMPT[0][1] == COMPT[1][1]:
    # la valeur majoritaire est nécessairement la dernière valeur : on la retourne
    return T[-1]
  # si il n'y a pas égalité de nb_a et nb_b pour les 2n premiers éléments du tableau : n
b_a >= n//2+1 OU nb_b >= n//2+1 : on a la valeur majoritaire
  elif COMPT[0][1] > COMPT[1][1]:
    return COMPT[0][0]
  else :
    return COMPT[1][0]
```

Complexité :

Nombre de comparaisons : on parcourt le tableau entre le deuxième et l'avant-dernier élément, et à chaque tour de boucle, on fait une comparaison : le nombre de comparaisons vaut donc

$$2n-1 = u_n - 2$$

À la fin, seules des comparaisons entre entiers sont faites (on ne les compte pas).

Correction :

À la fin de la boucle, on a dénombré :

- les éléments du tableau, sauf le dernier, qui sont égaux au premier élément, disons a : on note nb_a leur nombre
- les autres, hormis le dernier, qui sont égaux à b (la deuxième valeur possible) : on note nb_b leur nombre.

Trois cas de figure se présentent :

- **Cas 1** : $nb_a = nb_b$: le dernier élément fera donc pencher la balance : s'il vaut a (resp. b), a (resp. b) est majoritaire.
- **Cas 2** : $nb_a > nb_b$: alors quel que soit le dernier élément, il y a au moins $\lfloor n/2 \rfloor + 1$ éléments égaux à a dans le tableau, donc a est majoritaire.
- **Cas 3** : $nb_b > nb_a$: de même, b est majoritaire.

II. Un algorithme avec égalité uniquement

1.

Par récurrence sur $i \in [\![1, n]\!]$:

À l'étape i de la première phase :

- P_1 : les balles de la boîte ont la même couleur que la dernière balle posée sur l'étagère
- P_2 : il n'y a pas deux boules contiguës de la même couleur sur l'étagère.

Initialisation : Pour $i = 1$, c'est clair.

Hérédité : Si la propriété est acquise pour les étapes précédant strictement l'étape $i \in [\![2, n]\!]$: on retire une balle b du sac, et on note C la couleur de la dernière balle posée sur l'étagère.

- **Cas 1** : b est de couleur C : alors on met b dans la boîte, et :
  - les balles de la boîte avaient déjà toutes la même couleur C que la dernière balle posée sur l'étagère (par hypothèse de récurrence), et cela reste le cas avec b : P_1 reste vraie.
  - P_2 est vraie (par hypothèse de récurrence).
- **Cas 2** : si b est d'une couleur différente $C' \neq C$: on place b sur l'étagère, et on place une balle b' de la boîte sur l'étagère, si possible (i.e. si la boite n'est pas vide).
  Alors :
  - *Si la boîte n'est pas vide* : b', la dernière balle posée sur l'étagère, est de couleur C, de même que les autres balles de la boîte (par hypothèse de récurrence) : P_1 reste vraie.
  - *Sinon, si la boîte est vide* : P_1 est vraie (quantification existentielle sur l'ensemble vide).
  - la suite des couleurs des balles de l'étagère est de la forme : $C_1 \cdots C_{i-1} C C' C$, où $C_j \neq C_{j+1}$ pour tout $j \in [\![1, i-2]\!]$ par hypothèse de récurrence. Donc P_2 reste vraie (car $C' \neq C$).

Dans tous les cas : P_1 et P_2 restent vraies, et la propriété est démontrée par récurrence.

Toute couleur $C' \neq C$ ne peut être majoritaire, car en notant $l \leq n$ le nombre de boules sur l'étagère, le nombre de boules de couleur C' dans le sac nb'_C est égal au nombre d'occurrences de C' dans la suite des couleurs des boules sur l'étagère, car

- aucune balle de la boîte n'est de couleur C' : elles sont toutes de couleur C
- toutes les balles sont soit dans la boîte, soit sur l'étagère, par construction.

Donc, comme il n'y a pas deux balles contiguës de la même couleur sur l'étagère :

$$nb'_C \leq l/2 \leq n/2$$

Donc par contraposée : s'il y a une couleur majoritaire, c'est C.

2.

Lors la phase 2 :

Invariant : à la fin de l'étape i, il y a, dans la poubelle, autant de balles de couleur C que de balles de couleurs différentes de C.

Démonstration : par récurrence sur i

- **Initialisation** : pour $i = 1$: la première balle examinée est de couleur C (dernière balle sur l'étagère), donc le résultat est acquis sachant que la balle précédente n'est pas de la même couleur (démontré en 1.) et qu'elle est aussi jetée.
- **Hérédité** : on note b la dernière balle sur l'étagère, de couleur C'.
- Si $C' = C$:
  - *S'il reste au moins deux balles* : le résultat est acquis par hypothèse de récurrence, et sachant que la balle précédant b n'est pas de la même couleur (démontré en 1.) et qu'elle est aussi jetée.
  - *Sinon* : Par hypothèse de récurrence, et comme on place la dernière balle (de couleur C) dans la boîte : l'invariant est maintenu.
- Si $C' \neq C$:
  - *Si la boîte n'est pas vide* : le résultat est acquis par hypothèse de récurrence, et sachant que les balle de la boîte sont de couleur C (démontré en 1.).
  - *Sinon* : on arrête, et l'invariant est maintenu.

Dans tous les cas, l'invariant est maintenu : il est donc démontré par récurrence.

3.

La phase 2 se termine si :

- **Cas 1** : l'étagère est vide : dans ce cas, comme toutes les balles étaient, au début de la phase 2, sur l'étagère ou dans la boîte : toutes les balles sont dans la boîte ou dans la poubelle.
  Comme
  - il y a autant de balles de couleur C que de balles d'une autre couleur dans la poubelle (invariant)
  - les balles de la boîte sont de couleur C (démontré en 1.)

  alors C est majoritaire si, et seulement si la boîte contient au moins une balle (ce qui est annoncé par l'algorithme).

- **Cas 2** : la balle courante a une couleur différente de C et la boîte est vide.
  Alors comme toutes les balles étaient, au début de la phase 2, sur l'étagère ou dans la boîte : toutes les balles sont sur l'étagère ou dans la poubelle.
  Comme la balle courante est d'une couleur différente de C, et comme il n'y a pas deux balles contiguës de la même couleur sur l'étagère, il y a nécessairement plus (au sens large) de balles de couleur différente de C que de balles de couleur C sur l'étagère.
  De plus, il y a autant de balles de couleur C que de balles d'une autre couleur dans la poubelle (invariant).
  Donc il y a plus (au sens large) de balles de couleur différente de C que de balles de couleur C en tout, et C n'est pas majoritaire (ce qui est annoncé par l'algorithme).

Dans tous les cas, l'algorithme est bien correct.

4.

Sans la comparaison superflue au début de la phase 2, dans le pire des cas :

- **Pendant la phase 1** : si $n - 1$ comparaisons sont faites.

- **Pendant la phase 2** : dans le pire des cas, seule la dernière balle est de couleur C, et toutes les balles de l'étagère (sauf les deux dernières) sont enlevées une à une (engendrant pour chacune une comparaison avec C) : pour maximiser la complexité, en notant l le nombre de balles enlevées une à une sur l'étagère (et donc de couleur différente de C), on veut maximiser l, sachant que :
  - si n est impair :

$$l + (l - 1) \leq n - 2$$

(car il y a alors $l - 1$ balles dans la boîte (toutes les balles de l'étagère étant jetées avec une balle de la boîte, sauf la dernière))
Donc, au maximum :

$$l = \lfloor n/2 \rfloor$$

  - si n est pair :

$$l + l \leq n - 2$$

(car il y a alors l balles dans la boîte (toutes les balles de l'étagère étant jetées avec une balle de la boîte))
Donc, au maximum :

$$l = \lfloor n/2 \rfloor - 1$$

Donc le nombre exact de comparaisons effectuées par l'algorithme dans le pire des cas est :

$$\begin{cases} n + \lfloor n/2 \rfloor - 1 = n + \lceil n/2 \rceil - 2 \text{ si } n \text{ est impair} \\ n + \lfloor n/2 \rfloor - 2 = n + \lceil n/2 \rceil - 2 \text{ sinon} \end{cases}$$

Borne inférieure de l'algorithme de recherche

En cas de test d'égalité, 4 cas de figure se présentent :

1. Si i ou j sont dans les gradins, alors la réponse est non et l'amphithéâtre est inchangé.
2. Si i et j forment un binôme alors la réponse est non et l'amphithéâtre est inchangé.
3. Si i (resp. j) est dans un binôme et j (resp. i) n'est pas son partenaire, alors la réponse est non et i (resp. j) est envoyé dans les gradins alors que son partenaire forme un troupeau singleton.
4. Si i et j sont dans un même troupeau, alors la réponse est oui et l'amphithéâtre est inchangé.
5. Si i et j sont dans des troupeaux différents, alors l'action dépend de la valeur

$$d = B + t$$

i) Si $d > m$: les deux troupeaux sont des singletons (voir la question 1) alors la réponse est non et i, j forment un binôme.
ii) Si $d = m$: la réponse est oui et les troupeaux de i et j fusionnent.

1.

Seul le cas **5.i)** modifie la valeur de d : d y est décrémenté de 1.

Donc

> d ne croît jamais strictement.

De plus : Invariant : à l'étape k :

- P_1 : $d \geq m$
- P_2 : $d > m$ implique que tous les troupeaux sont des singletons.

Preuve par récurrence sur k :

- Au début, pour $k = 1$: c'est clair, car $d = n > m$ et tous les troupeaux sont des singletons.
- Pour $k \in \mathbb{N}$, si l'invariant est vrai à l'étape $k - 1$:

Seul le cas **5.i)** modifie la valeur de d, et le cas échéant : cela signifie qu'on avait $d > m$, et comme d est décrémenté de 1, on a toujours :

- $$\underbrace{d - 1}_{\text{nouvelle valeur de } B+t} \geq m$$

  donc P_1 reste vrai.
- Si $d - 1 > m$, alors on vérifie que les troupeaux sont toujours bien des singletons (par hypothèse de récurrence, et car on n'a que fusionné de troupeaux en un binôme). Donc P_2 reste vrai.

Par récurrence, l'invariant est prouvé.

2.

Soit

$$E \overset{\text{def}}{=} (T[i_k] \equiv_k T[j_k])_{1 \leq k \leq l}$$

une exécution partielle, et $a \overset{\text{def}}{=} \{I_1, \cdots, I_s\}$ (où $\{I_1, \cdots, I_s\}$ est une partition de $[\![1, n]\!]$) une affectation admissible par rapport à A_E.

Il vient donc que :

- **A** : chaque troupeau de A_E est inclus dans un bloc de a (car chaque troupeau est monochrome, par le cas de figure **4.**)
- **B** : chaque binôme $\{i, j\}$ vérifie :

  $$\exists u \neq v \in [\![1, s]\!] ; i \in I_u \wedge j \in I_v$$

  car chaque binôme est bicolore, d'après le cas de figure **2**.
- **C** : chaque élément i dans les gradins a une couleur différente (*i.e* : est dans un bloc différent) de tous les éléments auxquels il a été comparés dans E, par les cas de figure **1** et **3**.

(*NB*: ces conditions nécessaires d'admissibilité sont aussi suffisantes)

Soit $k \in [\![1, l]\!]$ tel que $T[i_k] \equiv_k T[j_k]$

- *Si* $\equiv_k$ *est* $=$: alors
  1. Si i_k ou j_k sont dans les gradins : alors la réponse est non, et comme on n'échappe plus aux gradins ultérieurement, c'est le résultat obtenu pour a, avec **C**.
  2. Si i_k et j_k forment un binôme : alors la réponse est non, et comme i_k et j_k sont tel que :
  - soit ils sont restés ce binôme dans A_E

    - soit l'un des deux est dans les gradins, dans A_E
      c'est bien le résultat obtenu pour a, avec **B** et **C**.
3. Si i_k (resp. j_k) est dans un binôme et j_k (resp. i_k) n'est pas son partenaire : alors la réponse est non et i_k (resp. j_k) est envoyé dans les gradins, desquels il ne s'échappera plus (il est donc dans les gradins, dans A_E), ce qui correspond au résultat obtenu pour a, avec **C**.
4. Si i_k et j_k sont dans un même troupeau : alors la réponse est oui, et comme le troupeau courant dans lequel ils sont est :
    - soit inchangé dans A_E
    - soit fusionné avec un autre dans A_E
      ils restent dans le même troupeau, dans A_E, et c'est le résultat obtenu pour a, avec **A**.
5. Si i_k et j_k sont dans des troupeaux différents :
   i) Si $d > m$: alors la réponse est non et i_k, j_k forment un binôme. De même que dans le cas 2 : c'est bien le résultat obtenu pour a, avec **B** et **C**.
   ii) Si $d = m$: la réponse est oui et les troupeaux de i et j fusionnent. De même que dans le cas 4, ils sont toujours dans un même troupeau, dans A_E, donc c'est bien le résultat obtenu pour a, avec **A**.

- *Si $\equiv_k$ est $\neq$* : on procède de la même manière.

On a donc montré que a est cohérente avec E, et :

> Pour toute exécution partielle E, toute affectation a admissible par rapport à A_E, est cohérente par rapport à E.

Soit

$$E \stackrel{\text{def}}{=} (T[i_k] \equiv_k T[j_k])_{1 \leq k \leq l}$$

une exécution partielle : il suffit d'exhiber une affectation a admissible par rapport à A_E, pour montrer qu'il existe une affection cohérente avec E (a conviendra, par le lemme précédent).

En notant a une partition de $[\![1, n]\!]$:

$$\{I_1, \cdots, I_s\}$$

où :

- pour tout troupeau $\mathscr{T}$ de A_E, il existe $u \in [\![1, s]\!]$ tel que

$$\mathscr{T} = I_u$$

- Pour tout $i \in [\![1, n]\!]$, si i n'appartient pas à un troupeau, alors il existe $v \in [\![1, s]\!]$ tel que :

$$\{i\} = I_v$$

Alors est clairement admissible par rapport à A_E, par **A**, **B**, **C**.

Donc a est cohérente avec E, par le lemme précédent.

3.

Soit

$$E \stackrel{\text{def}}{=} (T[i_k] \equiv_k T[j_k])_{1 \leq k \leq l}$$

une exécution partielle telle que $d > m$ dans A_E.

Montrons qu'il n'existe pas de résultat r tel que (E, r) soit une exécution complète de l'algorithme.

Il suffit pour cela d'exhiber deux affectations a et a' cohérentes avec E telles que :

- a a une couleur majoritaire
- a' n'a pas de couleur majoritaire

Si A_E est de la forme :

$$\{\underbrace{\{\tau_1\}, \cdots, \{\tau_t\}}_{\text{troupeaux}}, \underbrace{\{b_{1,1}, b_{1,2}\}, \cdots, \{b_{B,1}, b_{B,2}\}}_{\text{binômes}}, \underbrace{G}_{\text{gradins}}\}$$

(*NB* : les troupeaux sont des singletons car $d > m$)

En posant :

Pour a	**Pour a'**
$\tau_1, \cdots, \tau_t \in I_{\mathcal{T}}$	
$\forall i \in [\![1, B]\!], b_{i,1} \in I_{\mathcal{T}}$	
$a \stackrel{\text{def}}{=} I_{\mathcal{T}} \cup \{\{b_{i,2}\}\}_{1 \leq i \leq B} \cup \{\{\gamma\}\}_{\gamma \in G}$	$a' \stackrel{\text{def}}{=} \{\{\tau_i\}\}_{1 \leq i \leq t} \cup \{\{b_{i,1}\}\}_{1 \leq i \leq B} \cup \{\{b_{i,2}\}\}_{1 \leq i \leq B} \cup \{\{\gamma\}\}_{\gamma \in G}$

On vérifie que a et a' sont admissibles par rapport à A_E

- a' l'est clairement, car les troupeaux sont des singletons
- a l'est, car si i et j sont dans le même binôme, la seule comparaison qu'ils on subi est la comparaison de l'un avec l'autre, donc il est loisible de choisir que la couleur d'un élément de la paire est la couleur commune aux troupeaux, et que la couleur de l'autre élément est différente (il n'y aura pas de conflit possible avec les couleurs d'autres éléments).

Or

- pour a : $|I_{\mathcal{T}}| = d \geq m$, donc la couleur C des éléments de $I_{\mathcal{T}}$ est majoritaire, et
$$r = C$$
- pour a' : les blocs sont de cardinal strictement inférieur à m et de couleurs différentes, donc aucune couleur n'est majoritaire, et
$$r = null$$

Donc l'algorithme ne peut pas conclure.

4.

Par contraposée du lemme précédent, et avec le fait, en plus, que $d \geq m$ (cf. question 1): il vient qu'à la fin d'une exécution complète,

$$d = m$$

De plus, **par l'absurde** : s'il n'y avait pas qu'un seul troupeau dans l'arène, on arriverait à une contradiction du fait que l'exécution soit complète, de la même manière que dans la question

précédente (avec les affectations a et a'), donc

> à la fin d'une exécution complète, il ne peut y avoir qu'un troupeau dans l'arène, et $d = m$.

5.

- Chaque élément dans les gradins a subi au moins deux comparaisons.
- Chaque binôme a subi exactement une comparaison.

Donc le nombre de tests qui renvoient faux vaut au moins $2g + B$

- Chaque troupeau peut être vu comme un graphe connexe dont
  - les éléments sont les sommets
  - et où deux élément sont reliés par une arête si, et seulement si il ont subi une comparaison ensemble (leurs couleurs sont égales)

  Donc pour $i \in [\![1, Tr]\!]$, si t_i est le cardinal du i-ième troupeau, le nombre de comparaisons subies par ses éléments vaut au moins (par connexité)

  $$t_i - 1$$

  Par suite le nombre de comparaisons subies par tous les éléments de troupeau vaut au moins :

  $$\sum_{i=1}^{Tr} (t_i - 1) = t - Tr$$

Donc le nombre de tests qui renvoient vrai vaut au moins $t - Tr$

6.

À la fin d'une exécution complète :

- il n'y a qu'un troupeau dans l'arène donc $Tr = 1$, $B = 0$
- $d = m$, donc $t = m$
- $g = n - d = n - m$

Donc le nombre de tests est minoré par

$$\begin{aligned} 2g + B + t - Tr &= 2n - m - 1 \\ &= n + (n - \lfloor n/2 \rfloor) - 2 \\ &= n + \lceil n/2 \rceil - 2 \end{aligned}$$

7.

La complexité en pire cas optimale est obtenue !