Exercices de révision

Partie I. Valeur majoritaire d'un tableau

Question 1. S'il existait deux éléments majoritaires dans un tableau de longueur n, ce dernier contiendrait au moins $2(\lfloor n/2 \rfloor + 1)$ éléments. Or $2(\lfloor n/2 \rfloor + 1) > 2(n/2) = n$, ce qui ne se peut.

Question 2. $\mu \geqslant \lfloor n/2 \rfloor + 1 \iff \mu > \lfloor n/2 \rfloor$ (car μ est entier) $\iff \mu > n/2$ ($\lfloor n/2 \rfloor$ est le plus grand des entiers inférieurs ou égaux à n) $\iff 2\mu - n > 0$.

Question 3.

```
let est_majoritaire x a =
  let n = vect_length a and m = ref 0 in
  for k = 0 to n-1 do
    if a.(k) = x then m := !m + 1
  done ;
  2 * !m - n > 0 ;;
```

Question 4.

```
let est_majoritaire x a =
  let n = vect_length a in
  let rec aux m = function
    | k when k = n      -> 2 * m - n
    | k when a.(k) = x -> aux (m+1) (k+1)
    | k                 -> aux m (k+1)
   in aux 0 0 ;;
```

Question 5. Il s'agit de l'itération des suites définies par $c_0 = a_0$, $m_0 = 1$ et les relations :

$$\forall k \in [\![1, n-1]\!],\ (c_k, m_k) = \begin{cases} (c_{k-1}, m_{k-1} + 1) & \text{si } a_k = c_{k-1} \text{ ;} \\ (c_{k-1}, m_{k-1} - 1) & \text{si } a_k \neq c_{k-1} \text{ et } m_{k-1} > 1 \text{ ;} \\ (a_k, 1) & \text{si } a_k \neq c_{k-1} \text{ et } m_{k-1} = 1. \end{cases}$$

Question 6. Montrons par récurrence sur k que la marge de c_k dans $(a_0, \ldots, a_k)$ est inférieure ou égale à m_k et que la marge de tout autre élément inférieure ou égale à $1 - m_k$.

- C'est bien la cas pour $k = 0$ (il y a un seul élément et il est majoritaire).
- Si $k > 0$, supposons le résultat acquis au rang $k-1$.
  - Si $a_k = c_{k-1}$, la marge de cet élément augmente d'une unité et la marge des autres décroit d'autant. On a donc $\mu(c_k) = \mu(c_{k-1}) \leqslant m_{k-1} + 1 = m_k$ et $a_j \neq c_k \Rightarrow \mu(a_j) \leqslant 1 - m_{k-1} - 1 = 1 - m_k$. Le résultat reste acquis au rang k.
  - Si $a_k \neq c_{k-1}$ et $m_{k-1} > 1$, la marge de a_k augmente d'une unité, celle des autres éléments (dont c_{k-1}) décroit d'autant. On a donc $\mu(c_k) = \mu(c_{k-1}) \leqslant m_{k-1} - 1 = m_k$, $\mu(a_k) \leqslant 1 - m_{k-1} + 1 = 1 - m_k$ et $a_j \notin \{c_k, a_k\} \Rightarrow \mu(a_j) \leqslant 1 - m_{k-1} - 1 < 1 - m_k$. Le résultat reste acquis au rang k.
  - Si $a_k \neq c_{k-1}$ et $m_{k-1} = 1$, la marge de a_k augmente d'une unité, celle des autres éléments (dont c_{k-1}) décroit d'autant. On a donc $\mu(c_k) = \mu(a_k) \leqslant 1 - 1 + 1 = 1$, $\mu(c_{k-1}) \leqslant 1 - 1 = 0$ et $a_j \notin \{a_k, c_{k-1}\} \Rightarrow \mu(a_j) \leqslant 1 - 1 - 1 = -1 < 0$. Le résultat reste acquis au rang k.

Question 7. À l'issue de la fonction **candidat** on sait que tous les éléments à l'exception de celui retourné par cette fonction ont une marge inférieure ou égale à 0 et ne peuvent donc être majoritaires. Pour déterminer s'il existe un élément majoritaire, il suffit d'appliquer la fonction **est_majoritaire** au résultat de la fonction **candidat** :

```
let majorite a =
  let x = candidat a in
  if (est_majoritaire x a) then x else raise Not_found ;;
```

Question 8.

```
let candidat a =
  let n = vect_length a in
  let rec aux c m = function
    | k when k = n     -> c
    | k when a.(k) = c -> aux c (m+1) (k+1)
    | k when m > 1     -> aux c (m-1) (k+1)
    | k                -> aux a.(k) 1 (k+1)
   in aux a.(0) 1 1 ;;
```

Partie II. Analyse en moyenne

Question 1. Cette fonction réalise l'itération des suites définies par $u_0 = v_0 = a_0$ et les relations $u_i = \min(a_i, u_{i-1})$ et $v_i = \max(a_i, v_{i-1})$. À l'évidence on dispose des invariants $u_k = \min(a_0, \dots, a_k)$ et $v_k = \max(a_0, \dots, a_k)$.

Question 2.

```
let min_max a =
  let n = vect_length a in
  let rec aux u v = function
    | k when k = n     -> (u, v)
    | k when a.(i) < u -> aux a.(i) v (k+1)
    | k when v < a.(i) -> aux u a.(i) (k+1)
    | K                -> aux u v (k+1)
  in aux a.(0) a.(0) 1 ;;
```

Question 3. Chaque itération utilise une ou deux comparaisons, donc le nombre minimal de comparaisons effectuées est minoré par $n-1$. Ce minorant est atteint par exemple dans le cas d'une liste triée par ordre décroissant.
De même, le nombre maximal de comparaisons est majoré par $2n-2$. Ce majorant est atteint lorsque le tableau est trié par ordre strictement croissant.

Question 4. Lors de la i^{e} itération, deux cas sont possibles :

- si $i+1$ est un minimum local, une seule comparaison est effectuée ;
- si $i+1$ n'est pas un minimum local, deux comparaisons sont effectuées.

Notons que $i+1 \in [\![2, n]\!]$ donc le minimum local $j = 1$ ne doit pas être comptabilisé.
Si m désigne le nombre de minimums locaux, le nombre total de comparaisons effectuées est donc égal à :

$$(m-1) \times 1 + (n-m) \times 2 = 2n - m - 1.$$

Question 5. Toute permutation admet 1 comme minimum local, donc $u_{n,0} = 0$.
Si $\sigma \in \mathfrak{S}_n$ présente n minimums locaux c'est que σ est décroissante. Il n'y a qu'une seule permutation décroissante, donc $u_{n,n} = 1$.
Soit $k \in [\![1, n-1]\!]$ et $\sigma \in \mathfrak{S}_n$ présentant k minimums locaux. Posons $p = \sigma(n)$.

- Si $p = 1$, n est un minimum local ; il y a donc $k-1$ minimums locaux dans $[\![1, n-1]\!]$, et donc $u_{n-1,k-1}$ permutations de ce type.
- Si $p \neq 1$, n n'est pas un minimum local ; il y a donc k minimums locaux dans $[\![1, n-1]\!]$. Sachant que p peut prendre $n-1$ valeurs distinctes, il y a donc en tout $(n-1)u_{n-1,k}$ permutations de ce type.

D'où : $u_{n,k} = (n-1)u_{n-1,k} + u_{n-1,k-1}$.

Question 6. $\mathrm{P}_n = \mathrm{X}^n + \displaystyle\sum_{k=1}^{n-1} \Big((n-1)u_{n-1,k} + u_{n-1,k-1} \Big) \mathrm{X}^k = \mathrm{X}^n + (n-1)\mathrm{P}_{n-1} + \mathrm{X} \sum_{k=0}^{n-2} u_{n-1,k} \mathrm{X}^k = (\mathrm{X} + n - 1)\mathrm{P}_{n-1}$.
Sachant que $\mathrm{P}_1 = \mathrm{X}$, on obtient : $\mathrm{P}_n = \mathrm{X}(\mathrm{X}+1)\cdots(\mathrm{X}+n-2)(\mathrm{X}+n-1)$.
P_n est scindé à racines simples, donc $\dfrac{\mathrm{P}_n'}{\mathrm{P}_n} = \displaystyle\sum_{k=1}^{n} \frac{1}{\mathrm{X}+k-1}$.

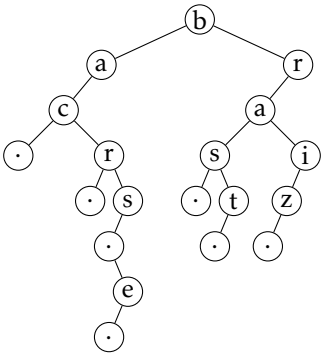
Question 7. Par définition, $\mathrm{M}_n = \dfrac{1}{n!}\displaystyle\sum_{k=1}^{n} k u_{n,k} = \dfrac{1}{n!}\mathrm{P}_n'(1) = \dfrac{\mathrm{P}_n(1)}{n!}\displaystyle\sum_{k=1}^{n}\dfrac{1}{k}$. Or $\mathrm{P}_n(1) = \displaystyle\sum_{k=1}^{n} u_{n,k} = \operatorname{card}\mathfrak{S}_n = n!$, donc $\mathrm{M}_n = \displaystyle\sum_{k=1}^{n}\dfrac{1}{k}$.

Question 8. Le nombre moyen de comparaisons effectuées par l'algorithme est donc égal à $2n - \mathrm{M}_n - 1 = 2n - \ln n + \gamma + o(1)$ qui est encore équivalent à $2n$, et donc bien plus grand que le $2n/3$ souhaité.

Partie III. Dictionnaire binaire

Question 1. Ajouter un point à la fin de chaque mot permet de faire en sorte qu'aucun mot ne soit préfixe d'un autre. Sans cette précaution il serait impossible que les mots bas et base coexistent dans le dictionnaire sous cette forme.

Question 2. La représentation sous forme d'arbre binaire de l'arbre donné en exemple est la suivante (le dictionnaire vide Nil n'est pas représenté) :



Question 3.

```
let chercher m a =
  let n = string_length m in
  let rec aux k = function
    | Nil                              -> false
    | Noeud ( . , _, _) when k = n     -> true
    | Noeud (_, _, a2)  when k = n     -> aux k a2
    | Noeud (c, a1, _) when m.[k] = c -> aux (k+1) a1
    | Noeud (_, _, a2)                 -> aux k a2
  in aux 0 a ;;
```

Question 4.

```
let branche m =
  let n = string_length m in
  let rec aux = function
    | k when k = n -> Noeud ( . , Nil, Nil)
    | k            -> Noeud (m.[k], aux (k+1), Nil)
  in aux 0 ;;
```

Question 5.

```
let inserer m a =
  let n = string_length m in
  let rec aux k = function
    | Nil                                  -> branche (sub_string m k (n-k))
    | Noeud ( . , a1, a2) as a when k = n -> a
    | Noeud (c, a1, a2) when k = n         -> Noeud (c, a1, aux k a2)
    | Noeud (c, a1, a2) when m.[k] = c     -> Noeud (c, aux (k+1) a1, a2)
    | Noeud (c, a1, a2)                    -> Noeud (c, a1, aux k a2)
  in aux 0 a ;;
```

Question 6.

```
let rec creer = function
  | []   -> Nil
  | t::q -> inserer t (creer q) ;;
```

Question 7.

```
let extraire a =
  let rec aux acc = function
    | Nil                -> []
    | Noeud ( . , _, a2) -> acc::(aux acc a2)
    | Noeud (c, a1, a2)  -> (aux (acc ^ (string_of_char c)) a1) @ (aux acc a2)
  in aux "" a ;;
```