Proposition de corrigé — sujet n°01 MPI Centrale-Supélec

Question 1. La fonction auxiliaire `est_prefixe` utilisée par `recherche` ne vérifie pas si l'indice `ind_part+ind_motif` dépasse la fin du tableau `part`. Ceci se produit si le motif est de taille plus grande que la partition ou bien si un préfixe strict du motif se trouve (pour la première fois) en fin de la partition.

Ceci peut se corriger en ajoutant, juste après le premier `if`, un test sur l'indice `ind_part` donné en paramètre :

```
let est_prefixe motif part ind_part =
  let rec parcours ind_motif =
    if ind_motif = Array.length motif then
      true
    (* correction: |motif| peut être > |part| *)
    else if ind_part = Array.length part then
      false
    else if motif.(ind_motif) = part.(ind_part+ind_motif) then
      parcours (ind_motif+1)
    else
      false
  in
  parcours 0
```

Question 2. Notons n la longueur de la partition et m la longueur du motif. La complexité de `est_prefixe` est en $O(m)$. Cette fonction est rappelée pour toutes les valeurs de `ind_part` allant de 0 à $n-1$. La complexité de `recherche` est en $O(mn)$. Les algorithmes de Boyer-Moore ou de Rabin et Karp qui sont au programme permettent d'améliorer cette complexité dans des cas favorables.

Question 3. Soit n la longueur de X et m la longueur de Y. L'ensemble de toutes les sous-séquences de X est de cardinal 2^n. Tester si une chaine W est une sous-séquence de Y se fait en temps $O(m)$ (on avance dans Y tant que l'on ne rencontre pas le premier caractère de W, puis on avance d'un caractère dans W et on recommence à partir de la position actuelle dans Y).

L'algorithme qui énumère toutes les sous-séquences de X donne alors une complexité en $O(m2^n)$. Le procédé étant encore valide en échangeant X et Y, on peut proposer une petite amélioration en comparant m et n pour proposer une complexité en $O(\max(n,m)2^{\min(n,m)})$.

Question 4. Le théorème précédent donne immédiatement :

$$c[i][j] = \begin{cases} 1 + c[i-1][j-1] & \text{si } x_i = y_j \\ \max(c[i-1][j], c[i,j-1] & \text{sinon} \end{cases}$$

Les cas de base sont $c[i][0] = 0$ et $c[0][j] = 0$.

On reconnait une stratégie de programmation dynamique.

Question 5. On propose cette fonction dans le fichier `plsc.ml`.

Question 6. La longueur attendue vaut 15.

Question 7. On peut procéder de deux façons :

— on modifie la matrice créée afin qu'elle stocke désormais un couple (l, dir) où l est la valeur calculée précédemment pour chaque case et $dir \in \{Haut, Gauche, Diag\}$ mémorise quel cas, dans la formule de la question 4, a permis de calculer $c[i][j]$;
— ou bien on remarque que l'on peut retrouver cette même information en partant de la case en bas à droite, en remontant vers la gauche ou vers le haut tant que l'on rencontre la même valeur. Lorsque l'on est en case (i, j) et que $c[i-1][j] < c[i][j]$ et $c[i][j-1] < c[i][j]$, alors cela signifie que le calcul de $c[i][j]$ provient du premier cas de la formule, donc $x_i = y_j$. On stocke cette lettre, on passe en case $c[i-1][j-1]$ et on recommence.

Cet algorithme s'exécute en temps $O(n+m)$.

Question 8. On propose cette fonction dans le fichier `plsc.ml`.

Question 9. On obtient la partition :

```
[|Re; Re; Fa; Mi; Sol; Re; Do; La; Sol; Fa; Mi; Do; La; Sol; Fa|]
```

Question 11. On propose la fonction suivante :

```
let count_occ (part : partition) : (note * int) list =
  let rec maj_note note occ_liste acc =
    match occ_liste with
    | [] -> (note, 1)::acc
    | (n, occ)::tl when n = note -> (n, occ+1)::(acc@tl)
    | hd::tl -> maj_note note tl (hd::acc)
  in
  let rec parcours_notes cpt occ_liste =
    if cpt = Array.length part then
      occ_liste
    else
      let n = part.(cpt) in
      parcours_notes (cpt+1) (maj_note n occ_liste [])
  in
  (* permet d'avoir un encodage même si la note n'apparaît pas dans
     la partition analysée *)
  let init = [(Do,1); (Re,1); (Mi,1); (Fa,1); (Sol,1); (La,1); (Si,1)] in
  parcours_notes 0 init
```

Question 12. Les commentaires du jury indiquent qu'il s'agit d'une question de cours, où l'on attend une discussion entre une implémentation naïve ou une implémentation par exemple avec des arbres en tas minimum.

Question 13. On propose ces fonctions dans `filePrio.ml`.

Question 14. On propose cette fonction dans le fichier `huffman.ml`.

Question 15. On propose cette fonction dans le fichier `huffman.ml`.

Question 16. On propose le test réalisé dans `tests_huffman.ml`.

Question 17. On propose cette fonction dans le fichier `huffman.ml`.

Question 18. On propose le test réalisé dans `tests_huffman.ml`.

Question 19. Cet aspect justifie l'utilisation de la variable `init` dans la fonction `count_occ`.

Question 23. On peut créer une table `titres` dont les colonnes sont `id_artiste` (clé étrangère vers `artistes.id`), `id_album` (clé étrangère vers `albums.id`), `nom` (chaine de caractères), `note` (nombre).

Question 24.

```sql
SELECT * FROM artistes WHERE debut >= 2000;

SELECT nom FROM artistes WHERE fin IS NOT NULL AND fin - debut + 1 >= 15;

SELECT albums.nom
  FROM albums JOIN artistes ON albums.id_artiste = artistes.id
  WHERE artistes.nom='Pink Floyd';

SELECT artistes.nom, albums.nom, albums.note
  FROM albums JOIN artistes ON albums.id_artiste = artistes.id
  WHERE albums.annee = 2001
  ORDER BY albums.note DESC;

SELECT AVG(note) FROM albums WHERE annee >= 1990 AND annee <= 2000;

SELECT COUNT(*), annee FROM albums WHERE annee >= 2005 GROUP BY annee;
```