



CONCOURS CENTRALE•SUPÉLEC

Sujet 01 — MPI

Représentation textuelle de musique




Durée : 3h

Centrale-Supélec

Préambule

Ce sujet d'oral d'informatique est à traiter, sauf mention contraire, en respectant l'ordre du document. Votre examinatrice ou votre examinateur peut vous proposer en cours d'épreuve de traiter une autre partie, afin d'évaluer au mieux vos compétences.

Le sujet comporte plusieurs types de questions. Les questions sont différenciées par une icône au début de leur intitulé :

- les questions marquées avec  nécessitent d'écrire un programme dans le langage demandé. Le jury sera attentif à la clarté du style de programmation, à la qualité du code produit et au fait qu'il compile et s'exécute correctement ;
- les questions marquées avec  sont des questions à préparer pour présenter la réponse à l'oral lors d'un passage de l'examinatrice ou l'examinateur. Sauf indication contraire, elles ne nécessitent pas d'appeler immédiatement l'examinatrice ou l'examinateur. Une fois la réponse préparée, vous pouvez aborder les questions suivantes ;
- les questions marquées avec  sont à rédiger sur une feuille, qui sera remise au jury en fin d'épreuve.


Votre examinatrice ou votre examinateur effectuera au cours de l'épreuve des passages fréquents pour suivre votre avancement. En cas de besoin, vous pouvez signaler que vous sollicitez explicitement son passage. Cette demande sera satisfaite en tenant également compte des contraintes d'évaluation des autres candidates et candidats.


1.3 Recherche de motifs

Soit $P = \langle p_1, p_2, \dots, p_m \rangle, p_i \in M$ une mélodie. On dit que P est un motif d'une partition $X = \langle x_1, x_2, \dots, x_n \rangle$ si $\exists i$ tel que $\forall k \in \llbracket 1; m \rrbracket, p_k = x_{i+k}$.

Par exemple, $\langle Do; Re \rangle$ est un motif de $\langle Do; Re; Mi \rangle$ mais pas de $\langle Re; Fa; Sol \rangle$.

Dans cette section, on propose de chercher un motif donné dans une partition.

▷ **Question 1.**  recherche : `Musique.partition -> Musique.partition -> int` dans `rech_naive.ml` est une implémentation d'un algorithme de recherche de motif. Implémenter un jeu de tests pour cette fonction, et corriger d'éventuelles erreurs d'implémentation.

▷ **Question 2.**  Décrire la fonction et discuter de sa complexité et des choix faits en terme de tests. Quel(s) algorithme(s) plus efficace(s) pourrait-on utiliser ?


1.4 Plus longue sous-séquence commune

On définit une sous-séquence d'une mélodie $X = \langle x_1, x_2, \dots, x_n \rangle$ comme une suite de notes $Z = \langle z_1, z_2, \dots, z_m \rangle$ telle qu'il existe une séquence strictement croissante $\langle i_1, i_2, \dots, i_k \rangle$ d'indices de X tels que $\forall j \in \llbracket 1; k \rrbracket, x_{i_j} = z_j$.

Par exemple, $\langle La; Sol; Do; La \rangle$ est une sous-séquence de $\langle La; Mi; Sol; Mi; Do; La; Mi \rangle$.

On cherche ici la plus longue sous-séquence commune (PLSC) à deux mots, soit la plus longue suite de notes qui est sous-séquence de X et de Y .

Une version naïve serait d'énumérer toutes les sous-séquences de l'un des mots, et de tester si ce sont des sous-séquences du second mot.

▷ **Question 3.**  Donner la complexité d'un tel algorithme.

Cependant, le problème de PLSC possède une propriété de sous-structure optimale, comme le montre le théorème ci-dessous.


Les classes naturelles de sous-problèmes correspondent à des paires de "préfixes" des deux séquences d'entrée. Plus précisément, étant donnée une séquence $X = \langle x_1, x_2, \dots, x_n \rangle$, on définit le i^{e} préfixe de X , pour $i \in \llbracket 0; n \rrbracket$, par $X_i = \langle x_1, x_2, \dots, x_i \rangle$.

Par exemple, si $X = \langle La; Mi; Sol; Mi; Do; La; Mi \rangle$, alors $X_4 = \langle La; Mi; Sol; Mi \rangle$ et X_0 représente la séquence vide.

Théorème 1 (Sous-structure optimale). Soient deux séquences $X = \langle x_1, x_2, \dots, x_n \rangle$ et $Y = \langle y_1, y_2, \dots, y_m \rangle$, et soit $Z = \langle z_1, z_2, \dots, z_k \rangle$ une PLSC quelconque de X et Y .

1. Si $x_n = y_m$ alors $z_k = x_n = y_m$ et Z_{k-1} est une PLSC de X_{n-1} et Y_{m-1} .
2. Si $x_n \neq y_m$ alors $z_k \neq x_n$ implique que Z est une PLSC de X_{n-1} et Y .
3. Si $x_n \neq y_m$ alors $z_k \neq y_m$ implique que Z est une PLSC de X et Y_{m-1} .

Source : *Algorithmique*, CORMEN, LEISERSON, RIVEST et STEIN, 3ème édition, chapitre 15.

▷ **Question 4.**  On considère qu'on a un tableau à deux dimensions $c[0..n][0..m]$, tel que $c[i][j]$ est la longueur des PLSC entre X_i et Y_j . Donner la formule permettant de calculer $c[i][j]$ et expliquer la stratégie.

Pour rappel, le module `Array` contient plusieurs fonctions permettant de manipuler des tableaux, et en particulier `make_matrix` : `int -> int -> 'a -> 'a array array`.

		0	1	2	3	4	5	6	7
		∅	Mi	Sol	Fa	La	Sol	Mi	Do
0	∅	0	0	0	0	0	0	0	0
1	Do	0	0	0	0	0	0	0	0
2	Mi	0	1	1	1	1	1	1	1
3	Fa	0	1	1	2	2	2	2	2
4	Si	0	1	1	2	2	2	2	2
5	La	0	1	1	2	3	3	3	3
6	Re	0	1	1	2	3	3	3	3
7	Sol	0	1	2	2	3	4	4	4

TABLE 1 – Exemple de tableau PLSC

▷ **Question 5.** ☞ Implémenter une fonction de prototype
`tabPLSC : Musique.partition -> Musique.partition -> int array array` qui crée, remplit et renvoie ce tableau pour deux partitions données.

Par exemple, si on prend `Musique.mus1 = [|Do ; Mi ; Fa ; Si ; La ; Re ; Sol|]` et `Musique.mus3 = [|Mi ; Sol ; Fa ; La ; Sol ; Mi ; Do|]`, on obtient le tableau 1. La PLSC est donc de longueur 4.

▷ **Question 6.** ⚡ Donner la longueur de la PLSC pour `Musique.mus5` et `Musique.mus6`.

On peut reconstruire la PLSC à partir du tableau, en remontant depuis la case $c[n][m]$. Par exemple, pour `Musique.mus1` et `Musique.mus3`, la PLSC est `[|Mi ; Fa ; La ; Sol|]`.

▷ **Question 7.** † Proposer un algorithme pour cela.

▷ **Question 8.** ☞ Implémenter une fonction de prototype
`trouvePLSC : Musique.partition -> Musique.partition -> Musique.note list` qui renvoie la PLSC de deux partitions.

▷ **Question 9.** ⚡ Donner la PLSC pour `Musique.mus5` et `Musique.mus6`.

1.5 Compression de Huffman

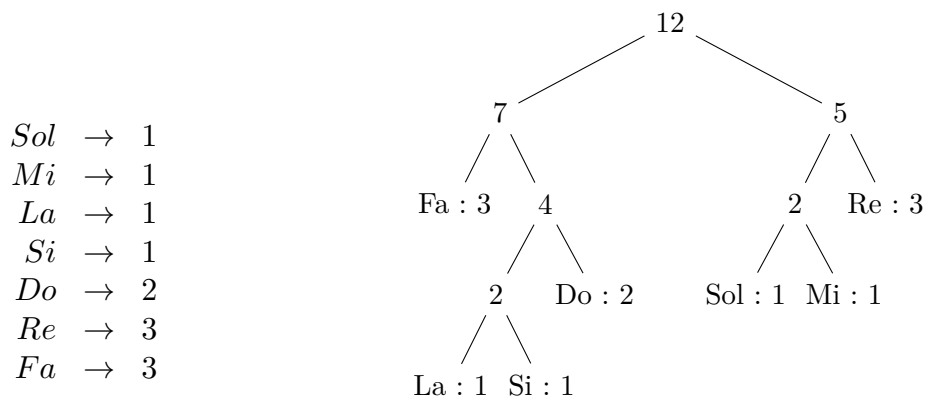
Pour transmettre ou stocker les partitions, on va vouloir les compresser.

On propose d'utiliser pour cela l'algorithme de Huffman, dont le but est de déterminer pour chaque caractère (ici note) un encodage, dont la longueur est grosso modo inversement proportionnelle au nombre de fois où apparaît ce caractère.

Par ailleurs, pour permettre la décompression, les encodages choisis par l'algorithme de Huffman sont tels qu'aucun encodage n'est préfixe d'un autre.

`Musique.m2` = [|Do ; Re ; Fa ; Sol ; Do ; Mi ; Fa ; La ; Si ; Fa ; Re ; Re|]

(a) Partition



(b) Nombre d'occurrences

(c) Arbre de Huffman

FIGURE 3 – Exemple d'application de l'algorithme de Huffman

▷ **Question 10.** ✎ Proposer un encodage pour les notes en fonction de l'arbre de Huffman de la figure 3.

▷ **Question 11.** 📖 Implémenter une fonction qui compte le nombre d'occurrences de chaque note dans une partition de prototype

```
compte_occ : Musique.partition -> (Musique.note * int) list.
```

▷ **Question 12.** † Pour dérouler l'algorithme de Huffman, on a besoin d'une file de priorité. Discuter les implémentations possibles d'une telle structure.

L'algorithme pour construire l'arbre de Huffman est indiqué ci-dessous :

```
filePrio = vide
pour chaque paire (note, occ)
    ajouter (occ, feuille(note)) a filePrio
tant que nombre d'éléments de filePrio > 1
    on extrait (p1, a1) et (p2, a2) les deux elements de priorite minimale
    nv_a = fusion(a1, a2)
    nv_p = p1+p2
    ajouter (nv_p, nv_a) a filePrio
renvoyer l'unique arbre contenu dans filePrio
```

▷ **Question 13.** 📖 Définir le type de votre file de priorité, avec comme priorité un entier, et comme valeur un élément du type `Arbre.arbre`, décrit dans `arbre.mli`. Implémenter les fonctions nécessaires à sa manipulation.

▷ **Question 14.** 📖 Implémenter l'algorithme de Huffman dans une fonction de prototype `algo_huffman : Musique.partition -> Arbre.arbre`.

▷ **Question 15.** 📖 Implémenter une fonction qui renvoie le dictionnaire d'encodage d'une partition donnée de prototype

```
dict_encodage : Musique.partition -> (Musique.note, string) Hashtbl.t.
```

Pour rappel, le module `Hashtbl.t` contient des fonctions permettant de manipuler une table de hachage.

▷ **Question 16.** ✎ Donner un dictionnaire d'encodage possible pour la partition

```
Musique.mus4 = [| Sol ; La ; Do ; Si ; Sol ; Fa ; La ; Si ; Sol ; La ;  
                Mi ; Do |]
```

▷ **Question 17.** ☞ Implémenter une fonction de prototype

```
encodage : Musique.partition -> (Musique.note, string) Hashtbl.t -> string
```

qui prend en paramètres une partition quelconque et un dictionnaire d'encodage, et renvoie une représentation textuelle de l'encodage.

▷ **Question 18.** ✎ Donner le résultat de l'encodage de `Musique.mus4` avec un dictionnaire construit à partir de `Musique.mus3`.

▷ **Question 19.** † Que se passe-t-il si on a une note que l'on essaie d'encoder qui n'était pas dans la partition de départ, et pour laquelle on n'a pas d'encodage ? Comment pourrait-on résoudre cela ?

▷ **Question 20.** ☞ Implémenter une fonction de prototype

```
decodage : string -> (Musique.note, string) Hashtbl.t -> Musique.partition)
```

qui, étant données une chaîne représentant une partition encodée, et un dictionnaire d'encodage, renvoie la partition originale.

En pratique, si on est en train de transmettre la partition compressée, il est probable que le destinataire n'ait pas l'arbre d'encodage. Il va donc falloir transformer l'arbre de Huffman en chaîne de caractères, pour le transmettre en amont du message.

▷ **Question 21.** † Expliquer comment on pourrait encoder l'arbre de Huffman.

▷ **Question 22.** ☞✎ Implémenter les fonctions nécessaires pour encoder l'arbre de Huffman, et décoder un message composé d'un arbre de Huffman suivi du message qu'il a servi à encoder. Écrire les prototypes des fonctions, et un court commentaire expliquant leur fonctionnement.

2 Une base de données musicale

2.1 Contexte

On a une base de données qui contient deux tables :

- une table contenant des artistes avec leur nom, la date de début de leur activité, et la date de fin (éventuellement `NULL`) (table 2) ;
- une table contenant des albums avec leur nom, l'identifiant de l'artiste correspondant, l'année de sortie et une note sur 10 (table 3).

id	nom	debut	fin
1	Nirvana	1987	1994
2	Pink Floyd	1965	1994
3	Daft Punk	1993	2012
4	The Doors	1965	1973
...			

TABLE 2 – Table des artistes

id	nom	id_artiste	annee	note
1	Nevermind	1	1991	7.8
2	The Dark Side of the Moon	2	1973	8.3
3	Random Access Memories	3	2013	6.7
4	Discovery	3	2001	7.8
...				

TABLE 3 – Table des albums

2.2 Description des fichiers fournis

Cette partie est accompagnée d'un fichier `setup_db.sql`, qui contient les requêtes SQL nécessaires à la mise en place de la base de données.

Pour lancer la base de données dans `sqlite3`, on doit utiliser les commandes suivantes :

```
$ sqlite3 <nom_au_choix>.db
...
sqlite> .read setup_db.sql
```

2.3 Modification de la base de données

▷ **Question 23.** † On souhaite ajouter à la base de données l'information des titres, avec un artiste, un album, un nom et une note. Décrire la table que l'on devrait rajouter, et son contenu.

2.4 Requêtes dans la base de données

▷ **Question 24.** ⚡ Récupérer, à l'aide de requêtes SQL, les informations ci-dessous. On indiquera sur la fiche réponse la requête et son résultat.

1. les attributs des artistes dont l'activité a commencé après 2000 ;
2. le nom des artistes dont l'activité est finie et a duré au moins 15 ans ;
3. le nom des albums de Pink Floyd ;
4. le nom de l'artiste, le nom de l'album, et la note de celui-ci, pour les albums de 2001 triés par note ;
5. la note moyenne des albums sortis entre 1990 et 2000 (inclus) ;
6. le nombre d'albums par année, à partir de 2005.