méthode Shazam

Il s'agit d'étudier une version simple de la **méthode Shazam**, qui permet d'identifier une musique à partir de son signal sonore. La méthode, basée sur une structure de données efficace, peut soit apprendre une nouvelle musique (stockage), soit rechercher une musique parmi celles apprises (requête). L'objectif sera de programmer un élément de base de cette méthode et d'étudier le code existant.

La méthode prend en entrée un son au format WAV (mono, échantillonné à 44100 Hz, sur 16 bits).

Le fichier à exécuter est `shapy.py` (auquel il faut rajouter le *shebang*, où est spécifié le chemin vers l'exécutable de Python 3), et un exemple d'utilisation est fourni dans le fichier `essai`.

1 Spectrogramme

La première étape de la méthode consiste à passer en représentation spectrale, en utilisant la **transformée de Fourier** discrète *(Discrete Fourier Transform – DFT)*, définie ainsi :

$$S[m] = \sum_{n=0}^{N-1} s[n] \cdot e^{-2i\pi nm/N} \quad \text{pour} \quad 0 \leq m \leq N/2$$

Elle transforme le signal sonore $s[n]$ (où n est le temps) en spectre $S[m]$ (où m est la fréquence).

Question 1 : Quelle est la complexité en temps de la transformée de Fourier discrète ?

Il existe une version plus efficace de cette transformée : la transformée de Fourier rapide *(Fast Fourier Transform – FFT)*.

Question 2 : Programmer la transformée de Fourier discrete (fonction `dft()`) en langages C et Python, à partir du code fourni.

Question 3 : Quelle est la complexité en temps de la transformée de Fourier rapide ?

2 Pics spectraux

La seconde étape de la méthode consiste à extraire les maxima locaux du spectrogramme ou **pics spectraux**, en estimant pour chaque pic ses paramètres (temps et fréquence).

Les pics de chaque spectre sont stockés dans une liste, et chaque liste est mémorisée dans un `buffer`, qui est lui-même une liste.

Question 4 : Expliquer ce que fait la fonction `buffer_shift()`. Pourrait-on faire plus efficace ?

3 Paires de pics

La troisième étape consiste à regrouper les pics par **paires**. Pour cela, on va se déplacer dans le temps, et à chaque instant et pour chaque pic présent à cet instant, on cherche des pics dans le passé (dans un intervalle de temps de 0.5 s à 3 s) et proches en fréquence (dans un intervalle de fréquence de ± 350 Hz), et on forme autant de paires que de pics trouvés.

4 Tableau associatif

La dernière étape utilise un **tableau associatif à 3 niveaux** pour le stockage (ou la recherche) des paires de pics :

— le premier niveau indexé par la fréquence du premier pic de la paire,
— le deuxième niveau indexé par la fréquence du second pic de la paire,
— le troisième niveau indexé par la différence de temps entre les deux pics,

et dans ce tableau on stocke (pour chaque paire de pics) un couple :

(date du premier pic, numéro de la musique).

Dans le cas du stockage d'une musique, on rajoute un tel couple dans le tableau.

Dans le cas d'une recherche d'une musique, on récupère le (ou les) couples présent(s) dans le tableau à cet endroit, et pour chaque couple trouvé on accumule dans un histogramme portant le numéro de la musique (du couple trouvé) à l'indice correspondant à la différence entre la date du couple trouvé et la date correspondant à la requête.

Question 5 : Cette structure de données est manipulée par le module `table.py`. En analysant les fonctions `table_get()` et `table_set()`, expliquer comment fonctionne cette structure de données et donner la complexité pour insérer et rechercher un élément.

À la fin, le maximum global dans les histogrammes donne la musique identifiée (correspondant au numéro de l'histogramme où le maximum est trouvé), ainsi que le moment dans la musique identifiée (correspondant à l'indice où ce maximum est trouvé).

Question 6 : Quelle est la complexité de la fonction `histograms_max()` ?