Décembre 2018 Corentin Denis 58701700

Dinh Thanh Phong Do 13601700

Rapport projet informatique

- ## Limitations et problèmes connus du code

  Notre programme n'est pas parfaitement optimisé. En effet, dans beaucoup de fonctions, nous faisons un appel récursif à Mix (pour les filtres) et à PartitionToTimedList (pour les transformations).

  De plus, certaines de nos fonctions contiennent des variables locales qui ne sont pas indispensables et pas beaucoup utilisées (parfois une seule fois) mais qui facilitent le code. Ces variables prennent un espace « inutile » en mémoire. Ce qui augmente la complexité spatiale de notre programme.
  Ensuite, notre code est complétement déclaratif, ce qui est couteux en calcul et en espace.
  Notre code n'est pas « tail-recursive ».
  Certains problèmes n'ont peut-être pas été repérés du fait que nous n'avons pas assez testé nos fonctions.

- ## Construction non-déclaratives

  Nous n'utilisons pas de constructions non-déclaratives.

- ## Choix d'implémentation surprenants

  NoteToNumber et NumberToNote sont peut-être un peu compliqués à comprendre mais nous les avons commentés en conséquence.

- ## Complexité

  1. ### Duration

     Duration exécute 2 fonctions {Stretch Factor FlatPartition} et {SumOfNotes FlatPartition Acc} qui sont chacune de complexité n.
     Stretch est de complexité n (n étant la taille de FlatPartition) car Stretch doit parcourir toute la liste et donc faire n appels récursifs à {Stretch} (D'abord {Stretch FlatPartition.1} puis {Stretch FlatPartition.2.1}, etc.)
     Même raisonnement pour SumOfNotes.
     Duration a donc une complexité appartenant à O(n).

  2. ### Merge

     Complexité O(n) car Merge fait appel à Mult (O(n)) et à SumLine (O(n))

  3. ### Loop

     La fonction Loop est appelée une seule fois.
     Loop fait tout d'abord appel à {Len L} qui est de complexité O(n) (n étant la taille de la liste L) car Loop doit parcourir toute la liste et donc faire n appels récursifs à {Len}.
     Ensuite, elle fait appel à IntToFloat, FloatToInt et Floor qui sont chacunes des fonctions de complexité appartenant à O(1) car ces fonctions nécessitent un seul appel.

Loop renvoie ensuite un Append de 2 listes. {Append L1 L2} est de complexité O(n) (n étant la taille de L1) car cette fonction parcourt la première liste et colle la deuxième au bout. Les arguments de Append sont générés à l'aide des fonctions {Repeat Amount Samples} et {Cut StartFloat FinishFloat Samples}.
Si l'on prend une valeur très grande pour Amount (=n) et une taille très grande pour Samples (=n), on en déduit que Repeat est de complexité $O(n^2)$ car il y a n*n appels à Append.
Si FinishFloat = n, Cut est alors de complexité O(n).

En mettant tout cela ensemble, nous déduisons que Loop a une complexité de $O(n^2)$

- **Extension(s)**

Nous avons fait l'extension lissage.

Spécification : Dans le 1er tiers de la note, l'intensité augmente progressivement. Dans le 2ème, l'intensité reste constante et dans le 3ème, l'intensité redescend. Pour réaliser cette extension, nous avons simplement utilisé la fonction fade comme ceci : {Fade Duree/3.0 2*Duree/3.0 Samples}. Duree étant la durée de Samples.

Pour activer cette extension, nous avons mis un booléen dans la fonction {PartitionToSamples Partition Bool}. Il faut lui mettre la valeur « true » lors de son appel dans Mix (ligne 674 du code).