Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Tarea 2

IIC2133 - Estructuras de Datos y Algoritmos

Primer semestre, 2017

Entrega: Jueves 11 de mayo

Objetivos

- Convertir un problema en otro de manera de poder trabajarlo
- Investigar sobre estructuras de datos de uso común
- Abstraer el concepto de texto a secuencias de datos en un contexto específico

Introducción y Problema

Cuenta la leyenda que antiguamente existían melodías mágicas como una que al tocarla en una ocarina podía invocar a la lluvia. Con el tiempo se han ido perdiendo en el olvido historias como esta, y ya nadie sabe si existieron realmente.

El arqueólogo **Daphnes Nohansen** teorizó que, en realidad, cualquier melodía podía tener propiedades como las de la leyenda, pero para ello era necesario identificar su **alma**. Teniendo el alma de una melodía se podría tocar en un instrumento e invocar con esto los poderes de la melodía en su totalidad. **Nohansen** propuso que el alma de una melodía no era más que la sub-melodía que más se repetía dentro de esta, pero que la razón que dificultaba encontrarla era que podía estar restringida a una cantidad mínima de notas. El arqueólogo llamó al proceso de obtener el alma de una melodía como la **destilación** de esta, y murió antes de poder llevarlo a cabo.

Leyendo los estudios de **Nohansen**, te diste cuenta que este es un problema que requiere de potencia computacional: hacerlo a mano sería una locura. Es por eso que para esta tarea escribirás un programa que destile una melodía para obtener su **alma**.

Las melodías están representadas en formato MIDI, dado que este trabaja directamente con las notas y duraciones de cada instrumento por separado. Todo el manejo de MIDI está hecho por la librería de la tarea, por lo que debes preocuparte en formular la solución a partir de la representación simplificada que te será entregada. Se presenta a continuación una explicación de los distintos elementos que debes considerar para poder resolver el problema.

Una breve descripción de MIDI

Los archivos MIDI (.mid) consisten en un conjunto de *pistas*, cada una con un instrumento, que han de ser reproducidas simultáneamente. Estas detallan cuando un instrumento debe sonar o callar, y en que tono.

Una pista es en realidad un conjunto de *eventos*, los cuales suceden en un momento dado. En particular, nos interesan los eventos de **comienzo** y **término** de una nota.

Los eventos están dados por el momento en el que suceden y la nota a la que corresponden, además de datos que no interesan para este problema, como la intensidad de la nota, entre otros.

Estructura Musical

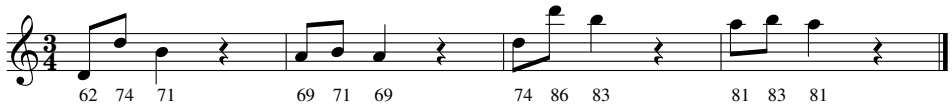
Melodías

Formalmente, se define una melodía τ como una serie de n *elementos musicales* secuenciales, cada uno con una duración relativa dada. Estos elementos pueden ser una nota o un silencio, donde las notas están representadas por números.

$$\tau = [\ (e_1, t(e_1)),\ \dots\ ,\ (e_n, t(e_n))\ ]$$

Donde e_i es el i-ésimo elemento de τ y $t(e_i)$ su duración relativa.

Por ejemplo, si tenemos la partitura[1]



Entonces $n = 16$, y

$$\tau = [\ (62,\ 1),\ (74,\ 1)\ ,(71,\ 2),\ (S,\ 2),\ \dots\ ,\ (81,\ 1),\ (83,\ 1)\ ,(81,\ 2),\ (S,\ 2)\ ]$$

Sub-Melodías

Una sub-melodía α de una melodía τ es simplemente una subsecuencia de esta, con un elemento inicial e_i y uno final e_j, con $i < j$. Esto también se puede escribir como $\tau[\ i : j\ ]$. Así,

$$\alpha = \tau[\ i : j\ ] = [\ (e_i, t(e_i)),\ (e_{i+1}, t(e_{i+1})),\ \dots\ ,\ (e_{j-1}, t(e_{j-1})),\ (e_j, t(e_j))\ ]$$

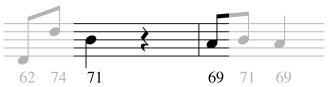
Si consideramos $m = |\alpha| = j - i + 1$, entonces podemos escribir α como sigue:

$$\alpha = [\ (\alpha_1, t(\alpha_1)),\ \dots\ ,\ (\alpha_m, t(\alpha_m))\ ]$$

Puede verse que una sub-melodía es también una melodía en si misma.

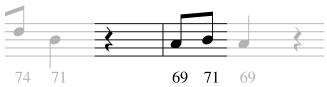
Por ejemplo, sub-melodías de la melodía τ del ejemplo anterior.

$\alpha_1 = \tau[\ 3 : 5\ ]$



$\alpha_1 = [\ (71,\ 2),\ (S,\ 2),\ (69,\ 1)\ ]$

$\alpha_1 = \tau[\ 4 : 6\ ]$



$\alpha_2 = [\ (S,\ 2),\ (69,\ 1)\ ,(71,\ 1)\ ]$

Una melodía α se dice *válida* cuando tanto α_1 como α_m son notas, y no silencios. Por ejemplo, del ejemplo anterior, α_1 es una melodía válida, mientras que α_2 no.

Notas

Se dijo que las notas se pueden expresar como números. Esto no es arbitrario. En realidad, entre la nota i y la nota $i + 1$ existe lo que se conoce como 1 *semitono* de diferencia. Esto es tal que para nuestro cerebro una diferencia de x semitonos entre dos notas es lo mismo independiente de las notas que sean.

Esto quiere decir que dos melodías pueden ser la misma sin tener las mismas notas, dado que dependerá de como la percibe nuestro cerebro.[2]. Esto depende directamente de los silencios, de la duración de los elementos, y de la diferencia entre las distintas notas de la secuencia. Si todos estos elementos son iguales, entonces las melodías son iguales.

[1] Por simplicidad, junto a cada partitura se incluirá el número asociado a cada nota.
[2] Esto es lo que se conoce como una misma melodía en distintas *escalas*

Equivalencia melódica

Primero se define la diferencia en *semitonos* entre dos notas x e y como

$$\Delta(x, y) = y - x$$

Se extiende esta definición para elementos; si el elemento e_i es un silencio, su valor como nota es el de e_{i-1}.

Nos interesa más que nada la diferencia entre un elemento k y el siguiente $(k+1)$ de una misma melodía α, la cual definimos como

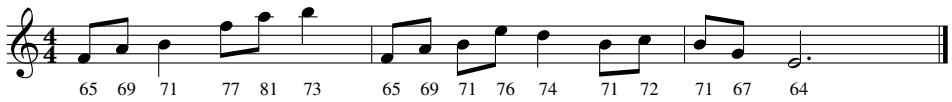
$$\partial(\alpha, k) = \Delta(\alpha_k, \alpha_{k+1})$$

Dos melodías válidas α y β son entonces *la misma* cuando se cumple lo siguiente:

- $|\alpha| = |\beta|$
- $t(\alpha_k) = t(\beta_k),\ \forall\ 1 \leq k \leq |\alpha|$
- $\partial(\alpha, k) = \partial(\beta, k),\ \forall\ 1 \leq k < |\alpha|$
- (α_k es nota) $\leftrightarrow$ (β_k es nota)[3], $\forall\ 1 \leq k \leq |\alpha|$

Esta propiedad se conoce como equivalencia y se denota como $\alpha \equiv \beta$

Por ejemplo, en la siguiente melodía τ:



- $\alpha_1 = \tau[\ 1:3\ ] \equiv \tau[\ 4:6\ ] \rightarrow$ se repite 2 veces
- $\alpha_2 = \tau[\ 1:2\ ] \equiv \tau[\ 4:5\ ] \equiv \tau[\ 7:8\ ] \rightarrow$ se repite 3 veces

Restricción de largo

Hay que considerar la restricción de largo. Este es un número entero $\mu > 1$ que indica si una melodía puede ser candidata a **alma** o no.

Debe cumplirse que si $\alpha = \tau[\ i:j\ ]$ es el alma de τ, entonces $|\alpha| \geq \mu$

En el ejemplo anterior, si $\mu = 2$, entonces el **alma** sería α_2. Si $\mu = 3$, entonces α_2 ya no sirve, por lo que el alma sería α_1.

Problema

Lo que debes hacer para esta tarea es escribir un programa en C que dado una pieza τ entregue la submelodía válida que más se repite dentro de τ, considerando además que esta debe tener un largo mayor o igual a un μ dado.

Para ello se recomienda revisar algoritmos de subsecuencias para Strings, y en particular acerca de la estructura de datos ***Suffix Array*** que permite resolver este problema en un tiempo eficiente.

Análisis

Deberás escribir un informe donde detalles tu implementación. En particular, se espera lo siguiente:

- Una descripción de como modelaste el problema. ¿Es esta una buena forma de hacerlo? Justifica. En caso de no serlo, explica como podría mejorar.
- Una descripción de las modificaciones que hayas hecho a la estructura de datos, tanto en construcción, como representación y funciones asociadas. ¿Qué te llevó a hacer esas modificaciones? ¿Cual es su aporte a la resolución del problema? Justifica.

[3]Esto quiere decir que para cada elemento de α, el elemento correspondiente en β es del mismo tipo: o ambos son nota, o ambos son silencio.

Input

Tu programa deberá recibir los siguientes parámetros:

1. La ruta del archivo que contiene la melodía a procesar
2. La cantidad mínima de notas que ha de tener el **alma** de la melodía
3. La ruta del archivo donde guardarás el alma de la melodía como una melodía en si misma

De la siguiente manera:

```
./distille input.mid mu output.mid [-b]
```

El archivo `input.mid` es un archivo MIDI que sigue el formato descrito anteriormente. Este representa el τ con el que deberás trabajar.

El número entero `mu` es el parámetro μ del problema.

El parámetro opcional `-b` es para el bonus. Si no implementas el bonus, haz que tu programa termine de inmediato al recibir ese parámetro. Si implementas el bonus entonces puedes ignorarlo.

Output

El output de tu programa es un archivo de audio `.mid`. Este archivo deberá describir correctamente la sub-melodía α de τ, $|\alpha| \geq \mu$, que más se repetía dentro de τ

Si quieres tener una noción de la correctitud de tu solución, puedes reproducir los archivos MIDI con un reproductor de música cualquiera.

Evaluación

La nota de tu tarea está descompuesta en distintas partes:

- 80 % corresponde a que las respuestas generadas sean correctas. Se evaluarán distintos casos de prueba.
- 20 % corresponde a la nota de tu informe.

Los tests tendrán un tiempo límite de 10 segundos c/u. Si tu programa no ha terminado en ese tiempo será cortado y tendrás 0 puntos en ese test.

Entrega

Deberás entregar tu tarea en el repositorio que se te será asignado; asegúrate de seguir la estructura inicial de éste.

Se espera que tu código compile con `make` dentro de la carpeta **Programa** y genere un ejecutable de nombre `distille` en esa misma carpeta. **No modifiques código fuera de la carpeta *src/distille*.**

Se espera que dentro de la carpeta **Informe** entregues tu informe en formato *PDF*, con el nombre *Informe.pdf*

Cada regla tiene asociado un puntaje, asegúrate de cumplirlas para no perder puntos.

Se recogerá el estado de la rama `master` de tu repositorio, 1 minuto pasadas las 23:59 horas del día de entrega. Recuerda dejar ahí la versión final de tu tarea. No se permitirá entregar tareas atrasadas.

Bonus

Como regla general, un bonus a tu nota se aplica solo si la nota correspondiente es $\geq$ 3,95. A continuación, las distintas formas de aumentar tu nota.

The real deal (Nota 10 en la tarea)

Para optar a este bonus, deberás adaptar tu solución para que considere que en realidad la música es más compleja que una secuencia de sonidos y silencios. En particular, una melodía **compleja** admite:

- Más de una nota simultánea
- Más de un instrumento, o capa de sonido

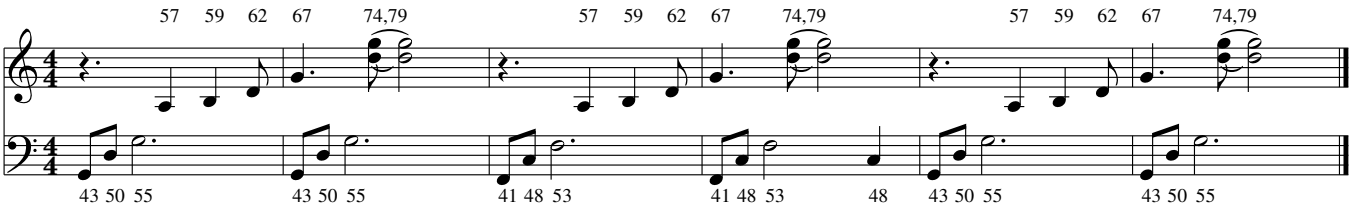
Para esto, se extiende la definición de equivalencia melódica a múltiples dimensiones:

Sea α una melodía **compleja** de $||\alpha||$ capas. La melodía de la capa $1 \leq k \leq ||\alpha||$ se escribe como α^j

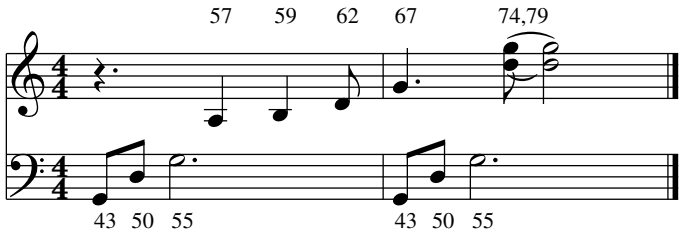
Dos melodías complejas α y β son equivalentes cuando se cumple lo siguiente:

- $||\alpha|| = ||\beta||$
- $\alpha^k \equiv \beta^k, \ \forall \ 1 \leq k \leq ||\alpha||$

Por ejemplo, si se tiene



La melodía que más se repite es



la cual se repite dos veces.

Para este bonus, tu programa deberá ser capaz de encontrar el **alma** de una melodía en caso de que esta sea una melodía **compleja**. Esta es, a su vez, una melodía compleja.

El input de tu tarea es el mismo. Puedes asumir que cuando se use el flag opcional `-b` de tu programa, entonces el archivo de input trae una melodía compleja.

Para esto, debes considerar que la modelación anterior de melodías en que se tiene una secuencia ordenada de notas y silencios ya no es válida para este problema, dado que mientras una nota está sonando ahora puede empezar a sonar otra nota en esa misma capa o en otra.

Por lo tanto deberás ver como procesar, quizas directamente, la estructura MIDI de la librería para poder obtener el **alma** de una melodía compleja.

Buen uso del espacio y del formato (+5 % a la nota de *Informe*)

La nota de tu informe aumentará en un 5 % si tu informe, a criterio del corrector, está bien presentado y usa el espacio y formato a favor de entregar la información.

Manejo de memoria perfecto (+5 % a la nota de *Código*)

Se aplicará este bonus si **valgrind** reporta en tu código 0 leaks y 0 errores de memoria, considerando que tu programa haga lo que tiene que hacer.

Requisitos de la librería

Para que el código base funcione deberás instalar la librería estándar de MIDI, SMF (Standard MIDI Files). Para esto, deberás escribir el siguiente comando en tu consola:

En **Ubuntu** y **Linux Subsystem for Windows (aka bash)**:

```
sudo apt-get install libsmf-dev
```

En **Mac OSX**:

```
brew install libsmf
```