Desplazamiento óptimo en la ejecución de una melodía para guitarra

Luis Carlos Quesada R. B65580

***Resumen*—Desarrollo de un método para optimización de distancia recorrida por la mano izquierda durante la ejecución de una melodía de guitarra, utilizando algoritmos exhaustivos, dinámicos y ávidos, de forma que retornan las posiciones en las que se debe tocar cada acorde para tocar la canción de la forma más óptima posible. Se implementan y prueban los algoritmos, por lo que se concluye que el algoritmo dinámico es más rápido que el exhaustivo, permite guardar datos útiles para próximas búsquedas y generar resultados bastante precisos indicando la cantidad de trastes que recorre la mano izquierda para ejecutar determinada melodía.**

I. Introducción

A la hora de ejecutar una melodía con una guitarra es necesario tocar un conjunto de acordes en orden. Cada acorde tiene al menos una forma de ser tocado, y para este proyecto un máximo de 3 formas o posiciones, produciendo así múltiples formas de ejecutar la misma melodía.

Las diferentes posiciones de un acorde, pueden resultar en una melodía mas compleja de ejecutar. De forma análoga es posible que se obtenga una forma mas sencilla de tocar la misma melodía. Para un estudiante principiante de guitarra, la ejecución de una canción debe ser lo más sencilla posible, así lograr su objetivo de aprender la melodía de forma rápida.

El objetivo de este proyecto es el desarrollo de una herramienta, la cual permita optimizar una melodía para guitarra, esto se lleva a cabo reduciendo el movimiento de la mano izquierda al máximo posible, para así facilitar la melodía a un guitarrista.

II. Metodología

Para la obtención de una melodía con movimiento de la mano izquierda reducido, se plantea realizar un algoritmo exhaustivo que encuentre la mejor combinación de posiciones en el acorde, también se lleva a cabo el desarrollo de un algoritmo dinámico que permita realizar la misma tarea de manera mas eficiente, finalmente se analiza la posibilidad de crear un algoritmo ávido para llevar a cabo la tarea, todos estos algoritmos realizados utilizando el libro *Introduction to Algorithms [1]* como referencia.

Cada algoritmo se analiza previamente a su programación, realizada en C++, el programa se compila en *Fedora 29*, utilizando un procesador Intel de 5 nucleos cuarta generación, 6Gb de memoria *RAM*. Para el proyecto se crea un tipo de datos llamado *Chord*, con las propiedades *label* de tipo *std::String* almacenando el nombre del acorde, y *Position* de tipo *float* almacenando las distancias de la posición en el acorde hasta el traste cero. Para la medición de distancias entre centroides se realiza de la siguiente forma:

Sean A_i, B_j centroides de los acordes A, B respectivamente, tal que la distancia entre ellos se defina como

$$|A_i - B_j| \tag{1}$$

III. Resultados

La búsqueda de la forma óptima se realizó con diversas técnicas de programación para encontrar ventajas a estas mismas.

III-A. Algoritmo Exhaustivo

Para realizar el algoritmo exhaustivo es necesario realizar un análisis del problema, para así crear una definición del algoritmo.

Definición del vector solución:

Sea $\vec{A} <A_0, A_1, ...A_i, ...A_n >$un vector de acordes que conforman la melodía a optimizar. sea $\vec{\sigma} <\sigma_0, \sigma_1, ...\sigma_i, ...\sigma_n >$, donde $\sigma_i \in \{1, 2, 3\}$, tal que σ_i indique la posición del acorde A_i a tocar, formando este el vector solución.

D**Definición del espacio:**

Como el espacio de posiciones esta definido por {1, 2, 3}, el espacio debe definirse como

$E = \{1, 2, 3\}_0 X \{1, 2, 3\}_1 X \{1, 2, 3\}_2 ... \{1, 2, 3\}_n$

lo que es igual a $\{1, 2, 3\}^n$, de esta forma definiendo también la cardinalidad del espacio como $|E| = 3^n$

Acotación del espacio: Es posible acotar el espacio si se ignora buscar en espacios que superan el mínimo global, pues como se define al final de la sección II, el espacio es continuamente creciente, por lo que no vale la pena realizar búsquedas superiores al tamaño mínimo actual, debido a que es difícil definir la cota, solo se puede saber que $|E'| \leq |E|$.

Cota asintótica:

La cota asintótica del algoritmo exhaustivo se encuentra de la siguiente forma:

$T(n) = 3T(n) + k$ con k como tiempo constante cualquiera, se resuelve la recurrencia:

$$\begin{aligned} T_1 &= k + 3T(n-1) = k + 3(k + 3T(n-2)) \\ T_2 &= 4k + 3(9T(n-2)) = 4k + 9(k + 3T(n-2)) \\ T_3 &= 13k + 27T(n-3) = 13k + 27(k + 3T(n-4)) \\ T(n) &= 3^n \end{aligned} \tag{2}$$

tal que el tiempo de ejecución es $O(3^n)$.

III-B. Algoritmo Dinámico

A continuación se realiza la implementación del algoritmo por el método de programación dinámica. **Determinación del oráculo:** Sea el oráculo una matriz $F[i][j]$ tal que $i = \{0, 1, ..., n\}$ y j = $\{0, 1, 2\}$ donde para cada entrada $F_i(j)$ sea la distancia mínima en $\{i, i+1, ...n\}$ tomando el centroide j. Ya que se utiliza un acorde fantasma, se toma como paso base que $F_n(j) = 0$ y como paso recursivo se toma

$$F_i(j) mn \begin{Bmatrix} |centroide[i+1][0] - centroide[i][j]| + F[i+1][0] \\ |centroide[i+1][1] - centroide[i][j]| + F[i+1][1] \\ |centroide[i+1][2] - centroide[i][j]| + F[i+1][2] \end{Bmatrix} \tag{3}$$

De forma similar, el valor objetivo se toma de la siguiente forma:

$$mn \begin{Bmatrix} F[0][0] \\ F[1][1] \\ F[0][2] \end{Bmatrix} \tag{4}$$

Cota asintótica: Para la cota asintótica del proceso dinámico es necesario tomar en cuenta el tiempo de ejecución de varios algoritmos, por lo que se tomará primero el tiempo del proceso de creación del oráculo [1].

$$\begin{aligned} T(n)\ &Para\ el\ ciclo\ externo \\ T(3)\ &Para\ el\ ciclo\ intermedio \\ T(2)\ &Para\ el\ ciclo\ interno \\ T(3)\ &Ciclo\ de\ bsqueda\ mnima \\ T(K)\ &lostiemposconstantes \end{aligned} \tag{5}$$

$$\begin{aligned} &Entonces \\ &T(n) * T(3) * T(2) + T(3) + T(K) \\ &T(6n + 3 + k) \\ &O(n) \end{aligned}$$

Para el algoritmo de búsqueda del camino de solución en la matriz Σ se considera el tiempo:

$$\begin{aligned} T(K)\ &Tiempos\ constantes \\ T(n-1)\ &Ciclo\ de\ bsqueda \\ T(3)\ &Ciclo\ mnimo\ inicial \end{aligned} \tag{6}$$

$$\begin{aligned} &EntoncesT(K) + T(n-1) + T(3) \\ &T(K + n - 2) \\ &O(n) \end{aligned}$$

III-C. Algoritmo Ávido

Para crear un algoritmo ávido es necesario poder mantener en cada iteración el hecho de que se obtiene el mejor resultado, sin embargo no es así, el mejor resultado de una etapa no garantiza que se mantenga como mejor resultado en la iteración siguiente, por lo que no es posible realizar un algoritmo ávido en este caso.

[1]Notese que para evitar sobrecarga del texto, se comprime los procesos constantes sin dependencia de ciclos como la letra K

IV. Conclusiones

Los algoritmos pueden retornar la forma más eficiente de tocar una melodía en guitarra; cuando la eficiencia es medida en distancia. Sin embargo, los algoritmos implementados difieren en comportamiento, un ejemplo de ello es el tiempo de ejecución. A la hora de medir cuanto tiempo tomaba el algoritmo exhaustivo para resolver un problema de poco mas de 15 elementos, los tiempos de ejecución eran terriblemente largos, incluso difíciles de medir, de modo contrario, para el algoritmo dinámico, la dificultad en su medición se da por la velocidad con la que se ejecuta, por lo que ningún caso de prueba retornó un tiempo medible por el *benchmark* realizado. En casos de pocos elementos ambos algoritmos retornan resultados exactamente iguales, lo que produce un sentimiento de confianza en el algoritmo, para pruebas de gran cantidad de elementos sin embargo, se hace imposible utilizar el algoritmo exhaustivo.

Una ventaja, además de la velocidad, que posee el algoritmo dinámico sobre el exhaustivo es la capacidad de guardar resultados. Durante el desarrollo, al momento de llenar la matriz de resultados Σ, una idea que surgió fue simplemente llenar con unos el numero menor, lo que resulto ser una idea errónea, pero esta idea dio inicio a la idea de que, con esta matriz se puede tomar el punto de inicio en la melodía desde cualquier acorde y con cualquier posición inicial, y aun así obtener el resultado más óptimo. Un punto a favor del algoritmo exhaustivo es lo fácil que es su implementación, requiriendo de poco tiempo para ser diseñado, pudiendo ahorrar tiempo de desarrollo y siendo una buena opción siempre y cuando se utilice en *datasets* muy pequeños.

V. Bibliografía

Referencias

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein (2009) *Introduction to Algorithms*, Massachussetts Institute of Technology.

Luis Carlos Quesada Estudiante de Bachillerato en Computación e informática de la Universidad de Costa Rica.