

El objetivo de esta sesión es evaluar la eficiencia de trabajar con heaps y estructuras de hash, comparándolas con los árboles de búsqueda binaria. Para este objetivo haremos una comparativa respecto al tiempo de ejecución y analizaremos sus ventajas y desventajas. La fecha límite de entrega de esta sesión es el día 02-06-2013. Documentar el código.

En esta sesión vamos a continuar con la aplicación de LastFM. En este caso queremos representar la información de todos los usuarios mediante á heaps y estructuras hash. La idea de la práctica es comparar un heap y una estructura de hash con un árbol de búsqueda binaria.

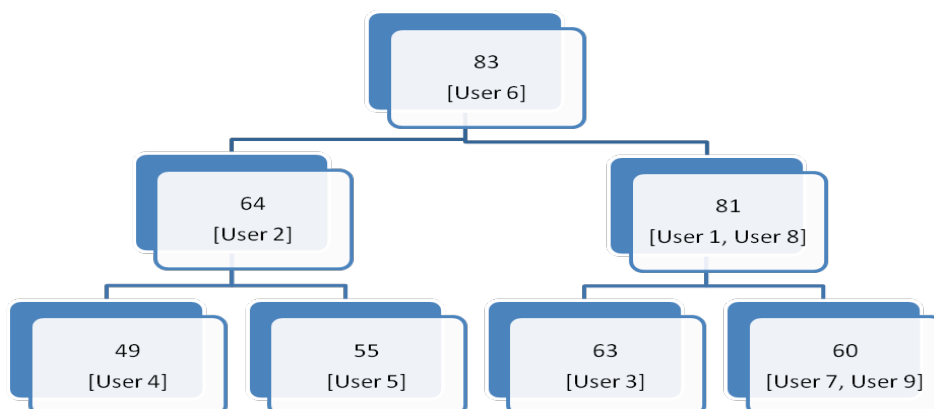
Recordad que tenemos toda la información de los usuarios en un fichero de datos llamado **LastFM_small.dat**. De este fichero vamos a extraer información de los usuarios.

Para implementar la aplicación seguir las instrucciones:

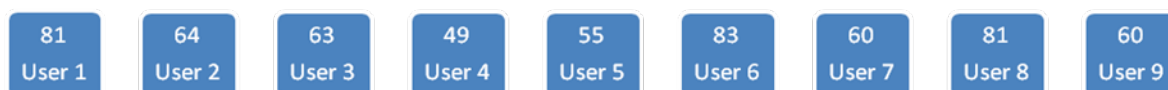
Paso 1. Crear un fichero **HEAP_nombre_apellidos.py** donde se definirá el heap y sus operaciones. Crear el heap teniendo en cuenta que el valor de relevancia del nodo padre siempre será mayor que el valor de relevancia de sus hijos.

Tened en cuenta que cada nodo del heap puede incluir una lista de usuarios con el mismo valor de relevancia; de este modo, al añadir un nuevo usuario debemos mirar si el nodo con la relevancia correspondiente existe. El funcionamiento deberá ser el siguiente: en el caso de no existir debemos crear el nodo y añadir el usuario correspondiente en la lista de este nodo, y en caso de que el nodo ya exista únicamente debemos añadir el usuario a la lista del nodo del heap que le corresponda según su relevancia.

En la siguiente figura tenéis un ejemplo de cómo quedaría el heap una vez insertados los primeros 9 datos del archivo *.dat*.

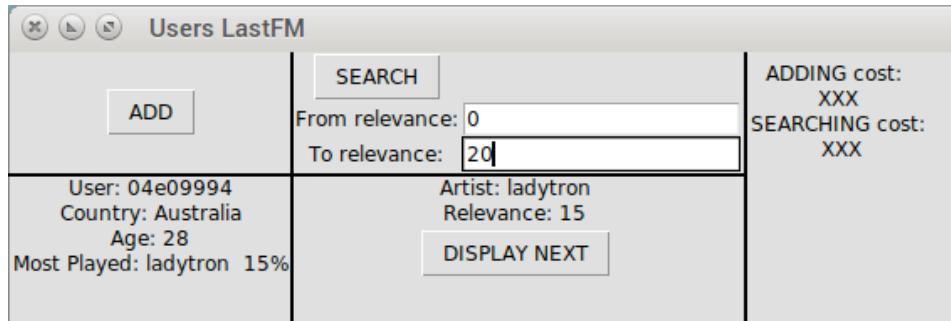


Notad que para la creación de este árbol los datos han sido introducidos en el siguiente orden:



Paso 2. Crear un fichero **nombre_apellido_S6.py** y leer el fichero LastFM_small.dat y guardarlo en el heap.

Paso 3. Crear un fichero **Nombre_Apellidos_HEAP_Interface.py** para implementar la interfície y funcionalidades solicitadas. Partiremos de la interfície de la práctica anterior, donde una posibilidad sería:



La interfície nos muestra “From relevance” y “To relevance” que sirven para definir el criterio de búsqueda de los usuarios. Cada vez que aprieta el botón “Search”, el programa debe calcular y enseñar el tiempo de ejecución de la búsqueda. Además, la pantalla nos muestra el tiempo computacional consumido para insertar los usuarios (ADDING cost, de 5000 en 5000 usuarios).

En esta ocasión, además queremos ver en la parte de estadísticas, la profundidad del heap.

Paso 4. Implementar la interfície de la aplicación en la clase **Nombre_Apellidos_HEAP_Interface.py** con las siguientes funcionalidades:

- ADD: añade 5000 usuarios del fichero en el heap. Cada vez que se apriete el ADD se añaden los 5000 siguientes usuarios hasta que se acabe el fichero. Recordad que se debe actualizar la información sobre la profundidad del heap en la interfície y el tiempo computacional consumido.
- SEARCH: dados dos valores “From relevance” y “To relevance”, recupere todos los usuarios que tienen una relevancia dentro de este intervalo.
- DISPLAY NEXT: muestra secuencialmente el id del usuario, su país, edad, genero, artista y relevancia.
- REMOVE: este botón eliminará del heap el nodo de máxima relevancia. En este caso también se debe actualizar la información sobre la profundidad del heap en la interfície y el tiempo computacional consumido.

Paso 5. Visualizar el tiempo de ejecución de la inserción (ADD) y la búsqueda (SEARCH) en la interfície.

Paso 6. Modificar la interfície de la práctica anterior **Nombre_Apellidos_ABB_Interface.py** para que se visualice la profundidad del árbol ABB cada vez que se pulsa el botón “ADD”.

Paso 7. Comparar el tiempo de ejecución promedio de insertar (ADD) y buscar (SEARCH) en el heap y en el árbol de búsqueda binaria (como mínimo 10 veces). Para esto podéis usar el fichero **LastFM_big.dat**.

Paso 8. Crear un fichero **HASH_nombre_apellidos.py** donde se definirá un árbol hash y sus operaciones. Para calcular la función de hash, utilizaremos la relevancia.

Paso 9. Crear un fichero **nombre_apellido_S6_hash.py** y leer el fichero LastFM_small.dat y guardarlo en el árbol hash.

Paso 10. Crear un fichero **Nombre_Apellidos_HASH_Interface.py** e implementar la interfície y funcionalidades solicitadas (las mismas que en los pasos 3 y 4).

Paso 11. Visualizar el tiempo de ejecución de la inserción (ADD) y la búsqueda (SEARCH) en el árbol hash.

Paso 12. Comparar el tiempo de ejecución promedio de insertar (ADD) y buscar (SEARCH) en el heap, en el árbol de búsqueda binaria y en el árbol hash (como mínimo 10 veces). Para esto podéis usar el fichero **LastFM_big.dat**.

NOTAS:

- **La nota va en función de los pasos que se realicen de la práctica. Los pasos de 1 a 7 tienen una nota máxima de 8, los pasos de 1 a 12 tienen una nota máxima de 10.**
- **La entrega de la sesión ha de ser únicamente por el Campus Virtual dentro del límite de tiempo predefinido.**
- **Entregad un fichero comprimido ("NombreApellidos_S6.zip") que contenga todo el código fuente (las tres aplicaciones).**
- **Recordad que es obligatorio estructurar bien la información, documentarla y argumentarla.**