Trabajo Especial Programación III

Informe

2da etapa

Fecha entrega: 19-05-16

Integrantes:

- Muñoz, Silvina
- Salvaneschi, Nicolas

Introducción

Para la resolución de esta segunda entrega debíamos elegir una estructura para mejorar la complejidad del algoritmo de búsqueda de usuarios.
En la etapa anterior esta complejidad era de O(n), osea que depende de la cantidad de usuarios que tenga el archivo sobre el que realiza dicha búsqueda.
Para llevar esta complejidad a *log n* nos decidimos por la estructura de arreglo ya que con un algoritmo de búsqueda binaria podríamos alcanzarla.

Análisis del comportamiento de la propuesta elegida

Como el objetivo de esta segunda entrega es mejorar los tiempos de búsqueda, como primer medida realizamos una inserción ordenada de los usuarios tanto en la precarga como en la alta.
Esto llevó a levantar altisimamente los tiempos de precarga y alta, pero a la vez lograr el objetivo de mejorar los tiempos de búsquedas ya que la lista de arreglo se encuentra ordenada.

buscarUsuario

El algoritmo de búsqueda particiona la lista arreglo ordenada dependiendo sobre qué mitad se sitúa el usuario pasado como parámetro(objetivo a buscar).
También se le pasa los índices correspondientes(min y max) y de esta forma, dependiendo por que lado de la mitad del arreglo base tenga que decidirse, se redefinen los índices y se llama a la misma función en forma recursiva.

En cuanto a los escenarios mencionados en la guia del TPE segunda entrega comentamos:

Punto (a) “Hay pocas altas de usuario y muchas verificaciones de existencia”.
Creemos que este planteo es más acorde a la implementación que realizamos en esta entrega. Como las altas son pocas y es lo que más tiempo lleva, las búsquedas se realizan rápidamente ya que el arreglo como dijimos se encuentra ordenado y nuestro método búsqueda recorta los tiempos de complejidad a (O(log n)).

Punto (b) “Hay muchas altas de usuarios y pocas verificaciones de existencia”.
Creemos que para realizar muchas altas la implementación que usamos es demasiado pesada, la potencialidad de nuestro planteo yace en la búsqueda y no en las altas como mencionamos en el punto anterior ya que el insertar ordenado tiene una complejidad n^2.
Para solucionar este planteo más bien se podría hacer una inserción desordenada en su totalidad, luego ordenarla con el algoritmo de mergeSort por ejemplo, y hacer la búsqueda en forma contraria al punto a, elemento por elemento, o se podría realizar la inserción

utilizando un algoritmo de búsqueda binaria y que este retorne el lugar donde posicionar el elemento.

Análisis de tiempo de ejecución

A continuación analizaremos los tiempos de ejecución de las altas y búsquedas respecto de la etapa 1 del trabajo.

Alta usuarios

Precarga	**Tiempo total Etapa 1**	**Tiempo total Etapa 2**
500mil	41527856	81560840993
1millon	50470965	185193949244

Podemos notar la diferencia de tiempos de ejecución durante la inserción en la etapa 2, dado que en el momento de insertar ordenado va comparando usuario por usuario, haciendo el corrimiento e insertandolo en el lugar correcto. Esto denota la complejidad O(n). La precarga de 3 millones no llegamos a tomar el tiempo debido a que a la tuvimos ejecutando varias horas y no finalizaba, es decir el tiempo que tarda en insertar es mayor al de la primer etapa.

Búsqueda usuarios

Precarga	**Tiempo total Etapa 1**	**Tiempo total Etapa 2**
500mil	211201693918	97456730
1millon	422932506395	112095546

En la búsqueda de usuarios podemos observar también la diferencia de tiempos de ejecución. El tiempo que tarda en hacer la búsqueda en la segunda etapa, llamando desde el método ‘existe()’, el cual llama al método ‘buscarUsuario()’ y busca de manera recursiva, hace que la búsqueda tenga una complejidad O(log n), y tenga una mejor performance a la hora de recorrer todo la lista.
Con las demás inserciones ocurre lo mismo que con las altas, dado a que no podemos hacer las altas, no tenemos los resultados de las búsquedas. Aunque ésta debe tardar mucho menos que en la primer etapa.

Cuestionario:

- ¿Será posible utilizar un árbol binario de búsqueda para el caso a) ?

Si sería posible utilizarlo, al estar ordenado lo que él busqueda binaria hace entre primera mitad y segunda mitad del arreglo, el árbol binario lo haría entre su lado izquierdo y derecho.

- ¿Sería conveniente hacerlo?

Seria practicamente lo mismo, lo unico que cambiaria es la estructura utilizada. La diferencia yace en los tiempos de inserción y los corrimientos en las altas de los usuarios.

- ¿Qué pros y contras le ve a esa solución?

Las mencionadas en el punto anterior, creemos que los problemas están en las inserciones de los nuevos usuarios.

- ¿Será posible encontrar alguna forma de garantizar que cuando cargamos los datos en una estructura de árbol, éste resulte en un árbol balanceado? Si fuera posible, describa cómo lo haría?

En un árbol balanceado la búsqueda es más eficiente, aunque al momento de insertar no nos asegura que el árbol permanezca balanceado, ya que el grado de balance depende del orden en que son insertados los nodos en el árbol.
Se puede balancear el árbol, realizando una rotación en el mismo.