Programación 3 - TUDAI 2017
Trabajo Práctico Especial - Etapa 2

Rodriguez, Juan Manuel - Coppis Javier

Problema

Etapa 2. Eficiencia Computacional

Hasta ahora, para verificar si existe un usuario debemos recorrer toda la estructura. En el peor caso, el tiempo insumido es proporcional a la cantidad de usuarios, o sea es O(n). Se pide reducir la complejidad computacional (el tiempo insumido) de esta operación al orden logarítmico (O(log n)) mediante la implementación de algún método que ordene la estructura, y otro de búsqueda logarítmica (búsqueda binaria u otra técnica que crea conveniente). Seleccione la estructura de representación de los usuarios y una estrategia que optimice la operación de verificación de existencia (búsqueda). Implemente estas operaciones siguiendo la propuesta elegida y justifique.

Analice el comportamiento de la propuesta elegida teniendo en cuenta los siguientes escenarios:

a) Hay pocas altas de usuarios y muchas verificaciones de existencia

b) Hay muchas altas de usuarios y pocas verificaciones de existencia

Se podría haber elegido otra alternativa de implementación que mejore algún escenario?

Utilizando los mismos archivos de entrada de datos y de operaciones, genere los archivos de salida (similar a la etapa 1).

Realice un informe donde se analicen los tiempos de ejecución y se comparen los comportamiento de las estructuras para los casos a y b, en función del tamaño de las entradas. Explicar cómo resolvería el caso a y el b, qué estructuras se deben usar, y algoritmos de búsqueda y ordenamiento, justificando la elección.

Sacar conclusiones.

Además, conteste en el informe las siguientes preguntas:

- Será posible utilizar un árbol binario de búsqueda para el caso a) ?
- Sería conveniente hacerlo?
- Qué pros y contras le ve a esa solución?
- 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?

Decisiones de diseño

Se decidió en primer instancia hacer una clase Usuario (nodo), con los atributos dni y gustos, la cual va a contener los datos de un usuario en cada instancia de lectura/escritura y contenida en cualquiera de los 3 contenedores pedidos.

Tenemos una interfaz “Lista”, adaptada a este problema, para así estandarizar la funcionalidad de las mismas.

Agregamos las clases “LeerDatos” y “EscribirDatos” que leen y escriben los archivos correspondientes colocando o leyendo los datos de la interfaz Lista, para no duplicar código. En la clase **LeerDatos** implementa los métodos:

CSVInsert: Tiene como parámetros de entrada PATH(dirección y nombre del archivo a leer), SEPARADOR(cadena de caracteres que lleva el carácter que separa las columnas del archivo csv) y DATOS (objeto del tipo GuardarDatos). Realiza la insercion de los datos (en esta etapa del archivo dataset_10000.csv) al contenedor.

CSVSeach: Sus parámetros son PATH, SEPARADOR y DATOS. Lee el archivo de búsqueda y por cada tupla busca en el contenedor pertinente (que contiene los datos del archivo de pre-carga + los de inserción).

CSVPreCharge: Recibe como parámetros PATH y SEPARADOR. Solo estos 2 porque lo único que hace este método es cargar el archivo csv a la estructura correspondiente.

Por otro lado en la clase **GuardarDatos**, tiene los métodos:

CSVWritter: Tiene como parámetro de entrada solamente SALIDA (cadena de caracteres con el nombre del archivo de salida) y su función es la de escribir cada tiempo, ya sea de búsqueda o inserción, en el archivo csv como una tupla, el dni y en el caso de la búsqueda agrega una columna más a la tupla para saber si ese usuario se encontró o no.

add: Agrega a un Vector, que tiene como atributo la clase, una cadena de caracteres que es la tupla correspondiente a una inserción o una búsqueda.

Agregamos en la primera linea de los csv búsqueda e inserción la tupla “dni;gusto1;gusto2;gusto3;gusto4;gusto5”, para estandarizar el formato de todos los archivos

Implementamos una búsqueda binaria por ser de orden O(log(n)). Esta es a lo que que el problema apuntaba para la optimización de los tiempos en las búsquedas y un quicksort para la parte de ordenamiento ya que también es de orden O(log(n)) por sobre un mergesort de orden O(n log(n)) o un burbuja O(n^2).

Hicimos la prueba insertando 10.000, ordenando, insertando 500.000, volviendo a ordenar y por último buscando 1.000.000 y 3.000.000 de usuarios sobre los 510.000 insertados.

Conclusión

Para la estructura de representación de los usuarios decidimos utilizar el ArrayList que habíamos implementado anteriormente, ya que tanto las búsquedas como los ordenamientos son más rápidos que con listas enlazadas, por estar indexadas.

Para la operación de búsqueda, nuestra optimización se realiza ordenando el ArrayList al finalizar la inserción de los usuarios.

Para el escenario donde hay pocas altas de usuarios y muchas verificaciones de existencia (a), la propuesta elegida nos parece la más acertada ya que la implementación fue realizada teniendo en cuenta este escenario.

Para el escenario donde hay muchas altas de usuarios y pocas verificaciones de existencia (b), la implementación podría ser optimizada cambiando el orden de operaciones para que el ordenamiento se realice antes de cada búsqueda en lugar de hacerlo después de cada inserción.

En conclusión para estos dos escenarios propuestos, la unica diferencia seria el momento en el cual se realiza el ordenamiento. Para el caso a, sería después de las inserciones y para el b antes de las búsquedas.

- Será posible utilizar un árbol binario de búsqueda para el caso a)?
  Sí, sería posible

- Sería conveniente hacerlo?
  Sí ya que no se tendría que usar ningún tipo de ordenamiento debido a que la estructura inserta siempre ordenada.

- Qué pros y contras le ve a esa solución?
  Pros: Sí el árbol binario fuera balanceado, se podría tener una métrica gracias al nivel máximo. Y por lo tanto las búsquedas son realmente de orden O(nivel máximo).
  Contras: La contra es que para que esté balanceado necesita reestructurarse con cada inserción.

- 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, escriba cómo lo haría?

Sí, para que un árbol binario se mantenga balanceado con cada inserción hay que buscar que los subárboles izquierdo y derecho sean balanceados y sus niveles no difieran en más de 1.