Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento Ciencia de la Computación
IIC3373 - Programación Concurrente

Tarea 2: Estructuras de datos concurrentes

Entrega: Miércoles 25 de Septiembre

Introducción:

El objetivo de esta entrega es el de familiarizar al alumno en la implementación de estructuras de datos ya conocidas con soporte para ambientes multi-threaded.

Descripción:

Usted deberá implementar un árbol AVL que soporte operaciones de inserción, deleción y búsqueda provenientes de múltiples threads distintos ejecutados en paralelo.

Para ello tenga en cuenta que su árbol debe realizar operaciones de rebalance (rotaciones) para evitar que la altura del árbol crezca demasiado.

No obstante, no es necesario que su árbol siga estrictamente las reglas de un árbol AVL. Es decir, es aceptable que existan periodos de tiempo limitados durante la implementación en que la diferencia de la altura de los dos subárboles de un nodo sea superior a 2. Nada más debe tener cuidado en demostrar que eventualmente, dado un tiempo de ejecución suficiente, el árbol logra adquirir una estructura en línea con las reglas de un árbol AVL.

Restricciones:

Debido al uso de un programa de testing para evaluar la tarea, su trabajo debe estrictamente estar desarrollado en el lenguaje JAVA.

Adicionalmente, no podrá hacer uso de ninguna librería third-party ni implementaciones no disponibles en el core del lenguaje.

Evaluación:

Para efectos de la evaluación, se usará un programa de testing automático al cuál los alumnos tendrán acceso desde la primera semana de publicado este enunciado. El programa utilizará su implementación junto con un archivo de testing y tiene dos posibles salidas: test exitoso o test fallido.

Para pasar un test el programa de testing efectuará un conjunto de operaciones de inserción, búsqueda y deleción definidos en un archivo de testing. El programa verificará que el resultado de las operaciones de búsqueda definidos en el archivo de testing sea consistente con las inserciones y deleciones previas.

Para poder hacer uso del programa de testing su código debe implementar la interfaz ISearchTree disponible dentro de los archivos descargables de la tarea.

Se utilizarán 5 tests distintos para evaluar su trabajo. Por cada test pasado completamente, usted recibirá un punto. No hay puntos parciales para esta parte de la evaluación. De estos 5 tests, 3 están disponibles dentro de los archivos adjuntos. Los 2 restantes se publicarán después de entregada la tarea.

Adicionalmente usted debe entregar un informe en donde explique su implementación y demuestre la correctitud de su algoritmo (es decir, que el resultado de cada operación sea correcto, que eventualmente su árbol adquiere la estructura de un AVL y que su algoritmo termina en un tiempo finito de tiempo para todo input posible). Este informe será calificado sobre 1 punto.

Bonus:

Esta tarea cuenta con dos posibles bonus de un punto cada uno a los cuáles usted puede acceder.

En primer lugar usted puede optar por implementar en vez de un árbol AVL, un árbol rojo-negro concurrente. Todas las reglas de este enunciado se aplicarían de forma análoga a esta implementación. En este caso, usted tendrá un punto base adicional por lo que su calificación será entre un 2 y un 8.

Adicionalmente se hará una competencia entre las implementaciones de todos los alumnos. El programa que opere más rápido recibirá un bonus de 1 punto, mientras que el segundo lugar recibirá un bonus de 0.5 puntos. Para ello se considerará la suma de los tiempos de todos los tests. Note que para poder optar a este bonus su programa debe también pasar todos los tests.

Plazos y entrega:

El plazo para entregar esta tarea es el día miércoles 25 de septiembre a las 23:59. Se descontará un punto de la nota final por cada hora o fracción de atraso.

Los archivos que usted debe entregar son:

- El código escrito. A lo menos usted debe entregar una clase que implemente la interfaz ISearchTree y todas sus dependencias.
- Un breve informe donde usted demuestre la correctitud de su algoritmo.
- Un archivo de texto .readme que explique cualquier supuesto que haya tomado en su implementación además de instrucciones para el ayudante y si acaso están optando al bonus de árbol rojo-negro (opcional).

Pueden enviar estos entregables comprimidos en un .zip o un .rar via mail a jibenedettoc@gmail.com.

En caso de problemas de envío por cualquier motivo, se aceptarán vías secundarias de entrega como por ejemplo compartir un Dropbox o subir su archivo a un servicio de File Sharing en línea (rapidshare, etc…) siempre y cuando se realice antes del plazo límite. Recuerden en este caso enviar un mail al ayudante con copia al profesor detallando los problemas que tuvieron al realizar el envío.

Referencias:

[1] http://www.blackbam.at/blackbams-blog/2012/05/04/avl-tree-implementation-in-java/

[2] Bronson, Nathan G., et al. "A practical concurrent binary search tree." *ACM Sigplan Notices*. Vol. 45. No. 5. ACM, 2010.

[3] L. Bouge, J. Gabarró, X. Messeguer, and N. Schabanel. Height-relaxed AVL rebalancing: A unified, fine-grained approach to concurrent dictionaries. ResearchReport RR1998-18, LIP, ENS Lyon, March 1998.

Anexo: Programa de testing

Para poder usted probar el programa de testing usted debe instanciar en la clase “Tester” línea 39 la variable “tree” (actualmente inicializada en null) con su árbol.

Al ejecutar entonces el programa se le pedirá que ingrese en consola el nombre del archivo de test a usar. Este debe encontrarse dentro del directorio de búsqueda del programa. Luego se comenzará a ejecutar el test.

Al finalizar la ejecución se imprimirá en consola si acaso el test tuvo éxito o no.

Anexo: Archivos de testing

A modo de referencia, los archivos de testing tienen el siguiente formato:

En la primera línea se indica el número N de threads que van a correr en paralelo. Todas las líneas que siguen contienen un comando cada una.

Cada comando tiene el formato:

<n° thread> <operación> <valor> <assert>?

<n°thread>: id entre 0 y N-1 que indica qué thread debe realizar la operación.

<operación>: string que indica si se trata de una inserción, búsqueda o deleción. Los posibles valores son FIND, INSERT, DELETE, SYNCHROINSERT, SYNCHROFIND y SYNCHRODELETE. Los tres últimos se utilizan para inserciones y deleciones que ocurren en threads distintos. Esto porque el programa de testing requiere elementos de sincronización para manejar correctamente estos casos. Con este mecanismo las demás operaciones son más eficientes.

<valor>: integer que indica la clave que buscar, insertar o eliminar.

<assert>: solo disponible para las operaciones de búsqueda, indica qué valor se espera del resultado de la operación. Sus valores posibles son true y false.