

Análisis del proyecto

Participantes:

Luis Ruiz Núñez
Guillermo Pichaco Panal
Pablo García Platero
Pablo Alarcón Carrión
Angela Roza Moreno

Clases del proyecto:

Clase [AvlTree](#):

- Esta clase implementa un árbol AVL, que es una estructura de datos de tipo árbol binario de búsqueda balanceada.
- Tiene un constructor que recibe un comparador para comparar los elementos del árbol.
- Tiene métodos para insertar, eliminar y buscar elementos en el árbol.
- Contiene métodos auxiliares para realizar operaciones como rotaciones, rebalanceo y cálculo de altura.
- La clase está bien comentada y sigue la licencia GNU Lesser General Public License.




Clase [AvlNode](#):

- Esta clase representa un nodo en el árbol AVL.
- Tiene atributos para almacenar el elemento, referencias a los nodos hijo y al padre, y la altura del nodo.
- Proporciona métodos para acceder y modificar estos atributos.
- También tiene métodos para verificar propiedades del nodo, como si es una hoja o si tiene solo un hijo.
- Es utilizada por la clase `AvlTree` para construir y manipular el árbol AVL.





En resumen, la clase `AvlTree` implementa el árbol AVL y utiliza la clase `AvlNode` para representar los nodos del árbol.

Cobertura de código de Jacoco:










































avlTree

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|------|---|------|--------|------|--------|-------|--------|---------|--------|---------|
|  avl |  | 97 % |  | 93 % | 8 | 104 | 5 | 237 | 1 | 48 | 0 | 2 |
| Total | 16 of 799 | 97 % | 7 of 111 | 93 % | 8 | 104 | 5 | 237 | 1 | 48 | 0 | 2 |












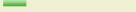
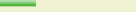














avl

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---------|---|-------|---|------|--------|------|--------|-------|--------|---------|--------|---------|
| AvlTree |  | 97 % |  | 92 % | 7 | 71 | 5 | 197 | 1 | 28 | 0 | 1 |
| AvlNode |  | 100 % |  | 96 % | 1 | 33 | 0 | 40 | 0 | 20 | 0 | 1 |
| Total | 16 of 799 | 97 % | 7 of 111 | 93 % | 8 | 104 | 5 | 237 | 1 | 48 | 0 | 2 |

AvlTree

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|------------------------------------|---|-------|---|-------|--------|------|--------|-------|--------|---------|
| insert(Object) |  | 0 % | | n/a | 1 | 1 | 3 | 3 | 1 | 1 |
| deleteNode(AvlNode) |  | 93 % |  | 78 % | 3 | 8 | 1 | 19 | 0 | 1 |
| searchNode(AvlNode) |  | 94 % |  | 91 % | 1 | 7 | 1 | 20 | 0 | 1 |
| rebalance(AvlNode) |  | 100 % |  | 100 % | 0 | 7 | 0 | 17 | 0 | 1 |
| searchClosestNode(AvlNode) |  | 100 % |  | 100 % | 0 | 7 | 0 | 23 | 0 | 1 |
| leftRotation(AvlNode) |  | 100 % |  | 100 % | 0 | 3 | 0 | 13 | 0 | 1 |
| rightRotation(AvlNode) |  | 100 % |  | 100 % | 0 | 3 | 0 | 13 | 0 | 1 |
| inOrder(AvlNode) |  | 100 % |  | 100 % | 0 | 2 | 0 | 6 | 0 | 1 |
| findSuccessor(AvlNode) |  | 100 % |  | 87 % | 1 | 5 | 0 | 10 | 0 | 1 |
| deleteLeafNode(AvlNode) |  | 100 % |  | 100 % | 0 | 3 | 0 | 8 | 0 | 1 |
| getBalance(AvlNode) |  | 100 % |  | 100 % | 0 | 3 | 0 | 7 | 0 | 1 |
| insertAvlNode(AvlNode) |  | 100 % |  | 80 % | 1 | 4 | 0 | 9 | 0 | 1 |
| deleteNodeWithALeftChild(AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 5 | 0 | 1 |
| deleteNodeWithARightChild(AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 5 | 0 | 1 |
| insertNodeLeft(AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| insertNodeRight(AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| height(AvlNode) |  | 100 % |  | 100 % | 0 | 2 | 0 | 5 | 0 | 1 |
| doubleLeftRotation(AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| doubleRightRotation(AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| AvlTree(Comparator) |  | 100 % | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| search(Object) |  | 100 % | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| compareNodes(AvlNode, AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| setTop(AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 3 | 0 | 1 |
| delete(Object) |  | 100 % | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| avlsEmpty() |  | 100 % |  | 100 % | 0 | 2 | 0 | 1 | 0 | 1 |
| toString() |  | 100 % | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| insertTop(AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| getTop() |  | 100 % | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 16 of 641 | 97 % | 6 of 85 | 92 % | 7 | 71 | 5 | 197 | 1 | 28 |

AvlNode

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|-------------------------|---|-------|---|-------|--------|------|--------|-------|--------|---------|
| updateHeight() |  | 100 % |  | 100 % | 0 | 5 | 0 | 8 | 0 | 1 |
| AvlNode(Object) |  | 100 % | | n/a | 0 | 1 | 0 | 8 | 0 | 1 |
| isLeaf() |  | 100 % |  | 100 % | 0 | 3 | 0 | 1 | 0 | 1 |
| hasOnlyALeftChild() |  | 100 % |  | 100 % | 0 | 3 | 0 | 1 | 0 | 1 |
| hasOnlyARightChild() |  | 100 % |  | 75 % | 1 | 3 | 0 | 1 | 0 | 1 |
| hasParent() |  | 100 % |  | 100 % | 0 | 2 | 0 | 1 | 0 | 1 |
| hasLeft() |  | 100 % |  | 100 % | 0 | 2 | 0 | 1 | 0 | 1 |
| hasRight() |  | 100 % |  | 100 % | 0 | 2 | 0 | 1 | 0 | 1 |
| setLeft(AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| setParent(AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| setRight(AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| setItem(Object) |  | 100 % | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| setHeight(int) |  | 100 % | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| setClosestNode(AvlNode) |  | 100 % | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| getLeft() |  | 100 % | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| getParent() |  | 100 % | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| getRight() |  | 100 % | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| getItem() |  | 100 % | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| getHeight() |  | 100 % | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| getClosestNode() |  | 100 % | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 0 of 158 | 100 % | 1 of 26 | 96 % | 1 | 33 | 0 | 40 | 0 | 20 |

En AvlTree

Podemos observar como el método "**delete()**" parece estar incompleto o con fallos. Esto puede deberse a que:

1. Se debe verificar si el nodo a eliminar es la raíz antes de intentar acceder a su padre.
2. Se debe verificar si el nodo a eliminar es una hoja antes de intentar acceder a sus hijos.
3. Después de eliminar el nodo, se debe actualizar la altura de los nodos en el camino de la raíz al nodo eliminado.

Luego el método "**searchNode(AvlNode)**" también tiene un error y es que debería buscar un nodo en el árbol AVL y devolverlo si lo encuentra, o null si no lo hace. Sin embargo, el método actualmente no busca ningún nodo y simplemente devuelve el nodo que se pasa como argumento.

Por otro lado, el método "**insert()**" no es testeado. Este método lo que hace es que a partir de un parámetro T, crea un nodo y llama al método "**insertAvlNode()**" con el nodo como parámetro.

Además, puede no funcionar correctamente si se intenta insertar un valor que ya se encuentra en el árbol. Es decir, no se está verificando si el valor a insertar ya está presente en el árbol y, por lo tanto, puede producirse una inserción duplicada.

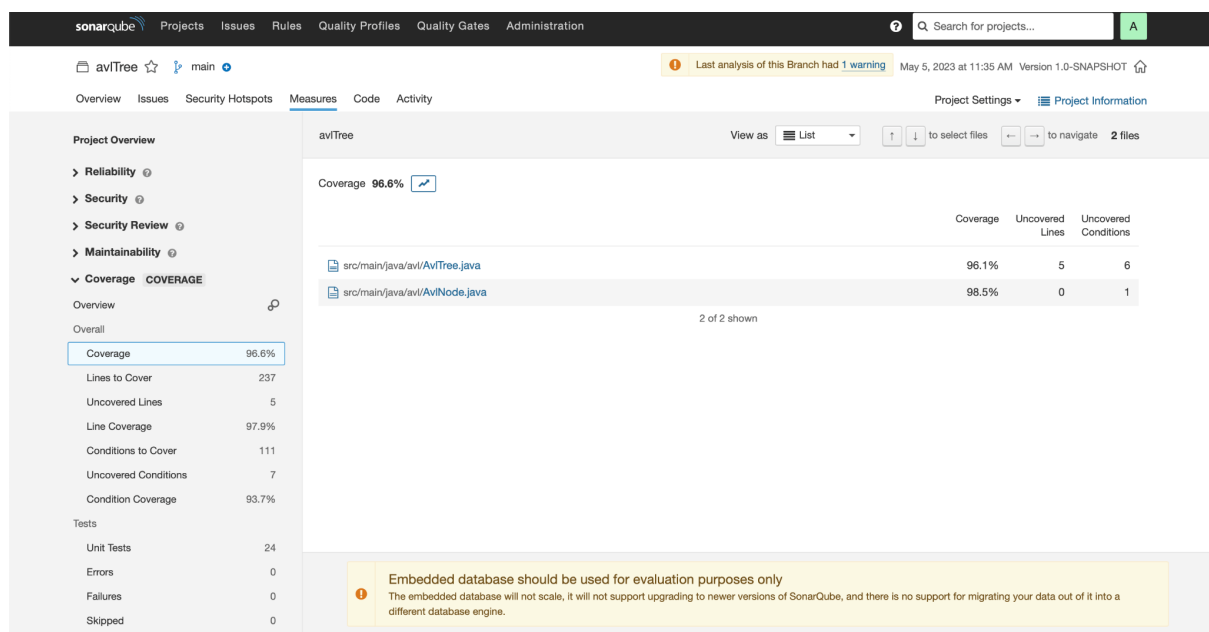
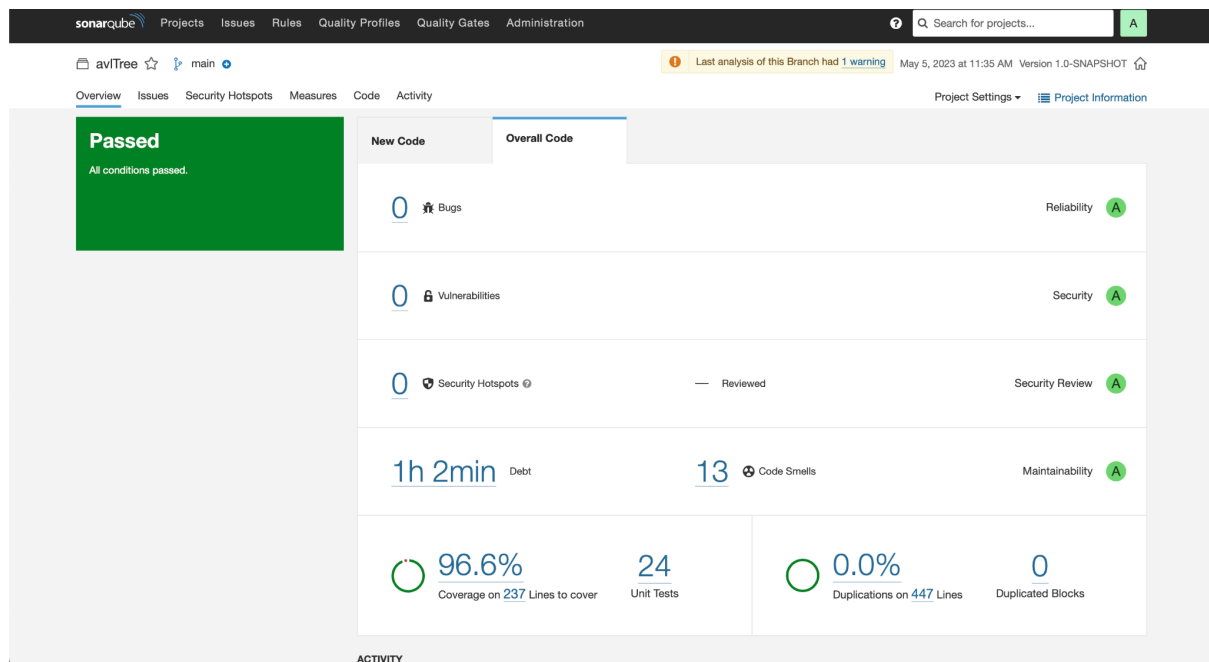
Si el árbol está vacío y se intenta insertar un nuevo nodo, puede haber problemas porque el método "**insert()**" no maneja este caso.

En AvlNode

El método "**updateHeight**" no verifica si hay hijos izquierdos y derechos antes de acceder a ellos. Entonces, se deben agregar verificaciones antes de acceder a los métodos "**getHeight()**" de los nodos izquierdo y derecho.

En los métodos "**hasOnlyALeftChild()**" y "**hasOnlyARightChild()**", el orden de las condiciones debe invertirse, ya que actualmente se devuelve "true" cuando el nodo tiene ambos hijos.

Calidad del código con Sonarqube:



Después de realizar el análisis de código con la herramienta Sonarqube, podemos ver que el código proporcionado no cuenta con bugs, vulnerabilidades, ni fallos en la seguridad del proyecto, sin embargo, el proyecto tiene 13 “code smells”, partes del código que podrían ser mejoradas.

El código tiene un coverage del 96.6% dejando 5 líneas sin cubrir de las 242 líneas de código, garantizando que el código no tiene fallos en esas líneas. Por lo que el código pasa todas las pruebas de Sonarqube.

Pruebas de Caja Negra:

Para las pruebas de caja negra, se puede considerar lo siguiente:

Pruebas de validez de entradas:

- Verificar el comportamiento de la aplicación cuando se le proporcionan datos de entrada válidos.
- Verificar el comportamiento de la aplicación cuando se le proporcionan datos de entrada inválidos, como valores nulos o vacíos.
- Verificar el comportamiento de la aplicación cuando se le proporcionan datos de entrada inesperados.

Pruebas de rendimiento:

- Verificar que el programa pueda manejar grandes cantidades de datos de entrada sin generar errores.
- Verificar que el programa pueda manejar grandes cantidades de datos de entrada dentro del tiempo de ejecución esperado.

Pruebas de funcionalidad:

- Verificar que la aplicación funciona según lo esperado, de acuerdo a las especificaciones del cliente o los requerimientos.
- Verificar que la aplicación produce los resultados correctos para un conjunto de entradas de prueba conocidas.
- Verificar que la aplicación maneja los casos límite de entrada de manera adecuada.

Ejemplos de pruebas de AVLNode

1. Prueba de caja negra para el método **"updateHeight()"**: se crea un nodo con algunos hijos y se comprueba que la altura se actualiza correctamente.
2. Prueba de caja negra para el método **"hasParent()"**: se crea un nodo y se comprueba que el método devuelve false.
3. Prueba de caja negra para el método **"hasLeft()"**: se crea un nodo con un hijo izquierdo y se comprueba que el método devuelve true.
4. Prueba de caja negra para el método **"hasRight()"**: se crea un nodo con un hijo derecho y se comprueba que el método devuelve true.
5. Prueba de caja negra para el método **"isLeaf()"**: se crea un nodo sin hijos y se comprueba que el método devuelve true.
6. Prueba de caja negra para el método **"hasOnlyALeftChild()"**: se crea un nodo con un hijo izquierdo y sin hijo derecho y se comprueba que el método devuelve true.

7. Prueba de caja negra para el método “**hasOnlyARightChild()**”: se crea un nodo con un hijo derecho y sin hijo izquierdo y se comprueba que el método devuelve true.

¿Tendría sentido además realizar pruebas de caja blanca?

En cuanto a si tiene sentido incluir pruebas de caja blanca, dependerá del caso. Dado que el código proporcionado parece estar diseñado para implementar un árbol AVL, podría ser beneficioso incluir algunas pruebas de caja blanca para verificar que la implementación del árbol AVL es correcta y que el código cubre todos los casos posibles. En este caso si tendría más sentido, ya que tenemos el código y podemos ver sus funciones, lo que hace que carezca de sentido realizar pruebas de caja negra y realizar los tests sin mirar el código del AVLTree, además, al realizar pruebas de caja blanca podemos garantizar que la cobertura del código sea del 100%

Algunas posibles pruebas de caja blanca incluyen:

- Verificar que la inserción y eliminación de nodos se manejan adecuadamente en todas las situaciones, incluyendo casos donde el árbol está desequilibrado.
- Verificar que la rotación de nodos se realiza adecuadamente en todas las situaciones, incluyendo casos donde el árbol está desequilibrado.
- Verificar que la altura del árbol se actualiza adecuadamente después de la inserción y eliminación de nodos.