|  |  |  |  |
| --- | --- | --- | --- |
| Clase: AVLTree | | Método: search | |
| Caso# | Descripción de la prueba | Estado inicial: | Resultado |
| 1 | Se busca un elemento en el árbol AVL. Para este caso se desea buscar los elementos 100, 1 y 55 que se supone deberían estar en el arbol | Un árbol previamente creado con los siguientes valores={5,4,1,8,23,10,0,  7,12,1000} | La prueba se ejecuta correctamente y los valores son encontrados en el árbol. |

|  |  |  |  |
| --- | --- | --- | --- |
| Clase: AVLTree | | Método: add | |
| Caso# | Descripción de la prueba | Estado inicial: | Resultado |
| 2 | Al árbol se agregan los valores que están creados en el stage2. Luego se crea un arreglo con los valores agregados al árbol, de tal manera que correspondan a los valores almacenados en el árbol recorrido por medio del recorrido inorden. | Un árbol AVL previamente creado, así como unas variables de tipo int que representan ciertos valores.(uno, dos, tres cuatro, cinco, diez; que efectivamente representan sus respectivos números) | La prueba se ejecuta sin problemas y efectivamente los dos arreglos(el creado y el devuelto por el método getDFS recorriendo el árbol por medio de inorden) son iguales. |

|  |  |  |  |
| --- | --- | --- | --- |
| Clase: AVLTree | | Método: remove | |
| Caso# | Descripción de la prueba | Estado inicial: | Resultado |
| 3 | Al árbol se agregan los valores que están creados en el stage2.Luego se procede a eliminar el “cinco” por medio del método remove. Y se crea un arreglo con los msmos valores agregados de tal manera que correspondan con el arreglo ideal que debería retornarse cuando se recorra el árbol por medio de preorden. | Un árbol AVL previamente creado, así como unas variables de tipo int que representan ciertos valores. (uno, dos, tres cuatro, cinco, diez; que efectivamente representan sus respectivos números) | La prueba se ejecuta sin problemas y efectivamente los dos arreglos(el creado y el devuelto por el método getDFS recorriendo el árbol por medio de preorden) son iguales. |

|  |  |  |  |
| --- | --- | --- | --- |
| Clase: AVLTree | | Método: clear | |
| Caso# | Descripción de la prueba | Estado inicial: | Resultado |
| 4 | Se prueba que el método clear efectivamente funciona, al comparar el tamaño del arreglo que representa el árbol después de que se ejecuta el método, con 0 | Un árbol previamente creado con los siguientes valores={5,4,1,8,23,10,0,  7,12,1000} | La prueba se ejecuta sin problemas, y efectivamente el tamaño del arreglo es 0, asegurando asi que la prueba funciona sin problemas. |

|  |  |  |  |
| --- | --- | --- | --- |
| Clase: RedBlackTree | | Método: search | |
| Caso# | Descripción de la prueba | Estado inicial: | Resultado |
| 1 | Se busca un elemento en el árbol rojinegro. Para este caso se desea buscar los elementos 233 y 734 que se supone deberían estar en el árbol | Un árbol previamente creado con los siguientes valores={8,4,34,98,233,1450,0,734,  1200,1000,98765,98754} | La prueba se ejecuta correctamente y los valores son encontrados en el árbol. |

|  |  |  |  |
| --- | --- | --- | --- |
| Clase: RedBlackTree | | Método: add | |
| Caso# | Descripción de la prueba | Estado inicial: | Resultado |
| 2 | Se compara la cantidad exacta de elementos que deberían estar agregadas en el árbol con la cantidad real que retorna el método size() | Un árbol previamente creado con los siguientes valores={8,4,34,98,233,1450,0,734,  1200,1000,98765,98754} | La prueba se ejecuta correctamente y los valores coinciden, demostrando así que el agregar funciona correctamente. |

|  |  |  |  |
| --- | --- | --- | --- |
| Clase: RedBlackTree | | Método: remove | |
| Caso# | Descripción de la prueba | Estado inicial: | Resultado |
| 3 | Se eliminan tres elementos del árbol y luego se procede a comparar si el tamaño de este coincide con el tamaño que debería tomar después de haber eliminado esos elementos, asi como a comprobar que cuando se buscan estos elementos en el árbol se retorna null. | Un árbol previamente creado con los siguientes valores={8,4,34,98,233,1450,0,734,  1200,1000,98765,98754} | La prueba se ejecuta correctamente y el tamaño deseado coincide con el tamaño real, tambieb cada prueba retorna su respectivo null cuando se busvan los elementos eliminados, comprobando así que el método remove funciona correctamente. |

|  |  |  |  |
| --- | --- | --- | --- |
| Clase: BVC\_APP | | Método: loadData(String, LocalDate, LocalDate | |
| Caso# | Descripción de la prueba | Estado inicial: | Resultado |
| 1 | Se prueba el cargue de datos a un árbol AVL.  Se envían una serie de datos al método loadData de tal manera que el árbol que va a ser instanciado es un AVLTree y se espera que se agreguen a este ultimo, 1191 elementos de los que se encuentran en el archivo USSPX500.txt | Se crea un objeto de tipo BVC\_APP y se llama al método LoadData con los siguientes datos:  *LocalDate start=LocalDate.of(2019, 3, 20);*  *LocalDate end=LocalDate.of(2019, 3, 21);*  *b.loadData("USSPX500", start, end);* | La prueba se ejecuta correctamente y efectivamente se comprueba que los 1191 elementos que se encuentran entre el rango de fechas establecido son los que se ha añadido al árbol avl |

|  |  |  |  |
| --- | --- | --- | --- |
| Clase: BVC\_APP | | Método: loadData(String, LocalDate, LocalDate | |
| Caso# | Descripción de la prueba | Estado inicial: | Resultado |
| 2 | Se prueba el correcto cargue de datos a un árbol AVL.  Se envían una serie de datos al método loadData de tal manera que el árbol que va a ser instanciado es un AVLTree y se espera que se agreguen a este ultimo, 1191 elementos de los que se encuentran en el archivo USSPX500.txt  Creamos un solo objeto de la clase FinancialThing el cual debería estar entre los datos cargados al árbol AVL en caso de que el cargue de datos sea correcto | Se crea un objeto de tipo BVC\_APP y se llama al método LoadData con los siguientes datos:  *LocalDate start=LocalDate.of(2019, 3, 20);*  *LocalDate end=LocalDate.of(2019, 3, 21);*  *b.loadData("USSPX500", start, end);*  Creamos un solo objeto de la clase FinancialThing con los siguientes datos:  *LocalDate ajam=LocalDate.of(2019, 3, 20);*  *FinancialThing webos=new FinancialThing(2, 2830.25, ajam);*  *Node<FinancialThing> eL=b.getStocks().search(webos);* | La prueba se ejecuta correctamente y se encuentra efectivamente que el elemento creado con los daos propuestos, se encuentra en el árbol AVL verificando asi el correcto cargue de los datos al árbol |

|  |  |  |  |
| --- | --- | --- | --- |
| Clase: BVC\_APP | | Método: loadData(String, LocalDate, LocalDate | |
| Caso# | Descripción de la prueba | Estado inicial: | Resultado |
| 2 | Se prueba el correcto cargue de datos a un árbol AVL.  Se envían una serie de datos al método loadData de tal manera que el árbol que va a ser instanciado es un AVLTree y se espera que se agreguen a este ultimo, 1191 elementos de los que se encuentran en el archivo USSPX500.txt  Creamos un solo objeto de la clase FinancialThing el cual debería estar entre los datos cargados al árbol AVL en caso de que el cargue de datos sea correcto | Se crea un objeto de tipo BVC\_APP y se llama al método LoadData con los siguientes datos:  *LocalDate start=LocalDate.of(2019, 3, 20);*  *LocalDate end=LocalDate.of(2019, 3, 21);*  *b.loadData("USSPX500", start, end);*  Creamos un solo objeto de la clase FinancialThing con los siguientes datos:  *LocalDate ajam=LocalDate.of(2019, 3, 20);*  *FinancialThing webos=new FinancialThing(2, 2830.25, ajam);*  *Node<FinancialThing> eL=b.getStocks().search(webos);* | La prueba se ejecuta correctamente y se encuentra efectivamente que el elemento creado con los daos propuestos, se encuentra en el árbol AVL verificando asi el correcto cargue de los datos al árbol |