|  |
| --- |
| **TAD ABB** |
| ABB { root, weight } |
| inv: |
| Operaciones básicas   * CreateABB → ABB * getRoot root x ABB → root * getWeight ABB → weight x Integer * getMax → Int * getMin → int * insert Node → ABB * modify Node x ABB → ABB * search value → Node * isInTree Key → Boolean * searchEqualTo Key → ArrayList<V> * searchLowerOrEqualThan Key → ArrayList<V> * searchLowerThan Key → ArrayList<V> * searchGreaterOrEqualThan Key → ArrayList<V> * searchGreaterThan Key →ArrayList<V> * delete Node x ABB → Node |

Operaciones

|  |
| --- |
| **CreateABB**  “Este metodo crea el arbol ABB vacio”  Pre:  Post: Se crea el ABB |
| **getRoot**  “Este método se encarga de dar la raíz del árbol”  Pre:El árbol ABB no es null  Post: |
| **GetWeight**  “Este método se encarga de dar el peso del árbol”  Pre:  Post: |
| **getMax**  “Este método retorna la llave mas grande del Arbol”  Pre: El árbol no es nulo  Post: |
| **getMin**  “Este método se encarga de retorna la llave de menor valor dentro del árbol”  Pre: arbolo no puede ser nulo  Post: |
| **Insert**  “Este método se encarga de insertar un nodo nuevo al árbol”  Pre: Nodo nuevo no es nulo  Post: Se agrega el nuevo nodo al arbol |
| **Modify**  “Este método se encarga de modificar un nodo existente”  Pre: El nodo modificado debe de existir  Post: Se modifica el nodo |
| **Search**  “Este método tiene la función de buscar una llave y retornar si este nodo existe junto la información que carga”  Pre:El nodo debe estar en el árbol, el nodo de búsqueda no puede ser null  Post: |
| **isInTree**  “Este método evalua si el nodo buscado esta o no en el árbol”  Pre: nodo distinto de null  Post: |
| **searchEqualTo**  “Este método busca un nodo con el valor de la llave asignada”  Pre: Llave no puede ser null, Arbol distinto de null  Post: |
| **searchLowerOrEqualThan**  “Este método busca en el rabol el nodo igual o menor a la llave que se ingresa”  Pre:Arbol no nulo y llave no nula  Post: |
| **searchLowerThan**  “Este método busca nodos con llaves menores a la que ha sido ingresada”  Pre: Arbol no nulo y llave no nula  Post: |
| **searchGreaterOrEqualThan**  “Este método se encarga de buscar nodos mayores o iguales a la llave ingresada”  Pre: Arbol no nulo, llave no nula  Post: |
| **searchGreaterThan**  “Este método se encarga de buscar llaves de nods que sean mayores a la ingresada”  Pre: Arbol no nulo, Llave no nula  Post: |
| **Delete**  “Este método se encarga de eliminar el nodo que se busca atraves de una llave”  Pre: Llave ingresada no nula, Arbol no nulo  Post: Se elimina el nodo del árbol |

**Diseños de casos de pruebas unitarias**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba 1: Verifica que el método insert añade exitosamente y cumple el invariante del árbol binario de búsqueda. | | | | |
| Clase | Método | Escenario | Entrada | Resultado |
| BinaryTree | +insert(K key, V value):void | Se ha creado un árbol binario | K=10  V=1 | El peso del árbol es 1 y el nodo raíz tiene key 10. |
| BinaryTree | +insert(K key, V value):void | Se ha creado un árbol binario y se la ha añadido un nodo con value 1 y key 10. | K=5  V=2 | El peso del árbol es 2 y el hijo izquierdo del nodo raíz tiene key 5.  El padre del nodo con value 2 es 1 |
| BinaryTree | +insert(K key, V value):void | Se ha creado un árbol binario y le se ha añadido lo siguientes nodos:  Key=10 y Value= 1  Key=5 y Value=2  En ese orden. | K=20  V=3 | El peso del árbol es 3 y el hijo derecho del nodo raíz tiene key 20 y value 3. El padre del nodo 3 es 1. |
| BinaryTree | +insert(K key, V value):void | Se ha creado un árbol binario y le se han añadido los nodos del escenario anterior más:  value=3  key=20 | K=6  V=4 | El peso del árbol es 4 y el hijo derecho del hijo izquierdo de la raíz tiene value 4.  El padre de 4 es 2. |
| BinaryTree | +insert(K key, V value):void | Se ha creado un árbol binario y se le se han añadido los nodos del escenario anterior más:  value=4  key=6 | K=5  V=5 | El peso del árbol es 5.  El hijo izquierdo del hijo izquierdo de la raíz tiene value 5.  El padre del hijo izquierdo del hijo izquierdo de la raíz es 2. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba 2: Verifica que el método search busca exitosamente un nodo en el árbol y retorna el valor de su value dado un key. | | | | |
| Clase | Método | Escenario | Entrada | Resultado |
| BinaryTree | +search(K): V | Se ha creado un árbol binario | K=10 | El método retorna null |
| BinaryTree | +search(K): V | Se ha creado un árbol binario y se la ha añadido un nodo con key= 10 y value= 1 | K=5 | El método retorna null |
| BinaryTree | +search(K): V | El mismo que el anterior | K= 10 | Retorna 1 |
| BinaryTree | +search(K): V | El mismo que el anterior y se la añade un nodo con key=5 y value=2 | K=6 | El método retorna null |
| BinaryTree | +search(K): V | El mismo que el anterior | K=5 | Retorna 2 |
| BinaryTree | +search(K): V | El mismo que el anterior y se le añade un nodo con key=20 y value 3 | K=20 | Retorna 3 |
| BinaryTree | +search(K): V | El mismo que el anterior y se le añaden dos nodos:  Key=6 Value=4  Key=15 Value=5  Key=25 Value=6 | K=25 | Retorna 6, es el mismo resultado que el método getMax |
| BinaryTree | +search(K): V | El mismo que el anterior | K=5 | Retorna 2, es el mismo resultado que el método getMin |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba 3: Verifica que el método delete elimina exitosamente un Nodo dado el key a eliminar. | | | | |
| Clase | Método | Escenario | Entrada | Resultado |
| BinaryTree | +delete(K): V | Se ha creado un árbol binario | K=10 | Retorna null |
| BinaryTree | +delete(K): V | Se ha creado un árbol binario y se la ha añadido un nodo con  Value=1  Key=10 | K=5 | Retorna null |
| BinaryTree | +delete(K): V | El mismo que el anterior | K = 10 | Retorna 1  El árbol está vacío. |
| BinaryTree | +delete(K): V | Se ha creado un árbol binario y le se ha añadido los siguientes nodos:  Key=10 value=1  Key=5 value=2 | K=5 | Retorna 2  La raíz es una hoja. |
| BinaryTree | +delete(K): V | El mismo que el anterior | K=10 | Retorna 1  La raíz tiene key 5 y es una hoja. |
| BinaryTree | +delete(K): V | El mismo que el anterior más:  Key=20 value=3 | K=20 | Retorna 3  El hijo derecho de la raíz es null. |
| BinaryTree | +delete(K): V | El mismo que el anterior y se le añaden dos nodos:  Key=6 Value=4  Key=15 Value=5  Key=25 Value=6 | K=20 | Retorna 3  El hijo derecho de la raíz tiene key 15 y su hijo derecho tiene key 25. |
| BinaryTree | +delete(K): V | El mismo que el anterior | K=5 | Retorna 2  El hijo izquierdo de la raíz es 6 y es una hoja. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba 4: Verifica que el método searchEqual busca exitosamente los nodos del árbol que sean iguales a un key dado. | | | | |
| Clase | Método | Escenario | Entrada | Resultado |
| BinaryTree | +searchEqualTo(K): V | Se ha creado un árbol binario | K=10 | El método retorna un ArrayList vacío. |
| BinaryTree | +searchEqualTo(K): V | Se ha creado un árbol binario y se la ha añadido un nodo con key= 10 y value= 1 | K=10 | El método retorna arrayList de size 1 que tiene en su posición 0 el value 1 |
| BinaryTree | +searchEqualTo(K): V | El mismo que el anterior y se le añaden los siguientes nodos:  Key=10 value=2  Key=10 value=3 | K= 10 | Retorna un arrayList de tamaño 3 donde:  Posición 0=1  Posición 1=2  Posición 2=3 |
| BinaryTree | +searchEqualTo(K): V | El mismo que el anterior y se la añade un nodo con key=5 y value=4 | K=5 | El método retorna un arraylist de size 1 que tiene en su posición 0 el value 4 |
| BinaryTree | +searchEqualTo(K): V | El mismo que el anterior | K=6 | Retorna null |
| BinaryTree | +searchEqualTo(K): V | El mismo que el anterior y se le añade un nodo con key=20 y value=5  Key=6 Value=6  Key=15 Value=7  Key=25 Value=8  Key=15 value=9 | K=20 | Retorna un arrayList de size 1 que tiene en su posición 0 el value 5 |
| BinaryTree | +searchEqualTo(K): V | El mismo que el anterior. | K=15 | Retorna un arrayList de size 2 donde:  Posición 0=7  Posición 1=9 |
| BinaryTree | +searchEqualTo(K): V | El mismo que el anterior. | K=25 | Retorna un arrayList de size 1 que tiene en su posición 0 el value 8 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba 5: Verifica que el método searchLowerOrEqualThan busca exitosamente los nodos del árbol que sean iguales o menores a un key dado. | | | | |
| Clase | Método | Escenario | Entrada | Resultado |
| BinaryTree | +searchLowerOrEqualTo(K): V | Se ha creado un árbol binario | K=10 | El método retorna un ArrayList vacío. |
| BinaryTree | +searchLowerOrEqualTo(K): V | Se ha creado un árbol binario y se la ha añadido un nodo con key= 10 y value= 1 | K=10 | El método retorna arrayList de size 1 que tiene en su posición 0 el value 1 |
| BinaryTree | +searchLowerOrEqualTo(K): V | El mismo que el anterior y se le añaden los siguientes nodos:  Key=10 value=2  Key=10 value=3 | K= 10 | Retorna un arrayList de tamaño 3 donde:  Posición 0=1  Posición 1=2  Posición 2=3 |
| BinaryTree | +searchLowerOrEqualTo(K): V | El mismo que el anterior y se la añade un nodo con key=5 y value=4 | K=10 | El método retorna un arraylist de size 4 donde:  Posición 0=1  Posición 1=2  Posición 2=3  Posición 3=4 |
| BinaryTree | +searchLowerOrEqualTo(K): V | El mismo que el anterior | K=6 | Retorna un arrayList de size 1 que tiene en su posición 0 el value 4 |
| BinaryTree | +searchLowerOrEqualTo(K): V | El mismo que el anterior | K=4 | Retorna un arrayList vacío |
| BinaryTree | +searchLowerOrEqualTo(K): V | El mismo que el anterior y se le añade un nodo con key=20 y value=5  Key=6 Value=6  Key=15 Value=7  Key=25 Value=8  Key=15 value=9 | K=20 | Retorna un arrayList de size 6 donde:  Posición 0=5  Posición 1=1  Posición 2=2  Posición 3=3  Posición 4=4  Posición 5=6 |
| BinaryTree | +searchLowerOrEqualTo(K): V | El mismo que el anterior. | K=15 | Retorna un arrayList de size 7 donde:  Posición 0=7  Posición 1=9  Posición 2=1  Posición 3=2  Posición 4=3  Posición 5=4  Posición 6=6 |
| BinaryTree | +searchLowerOrEqualTo(K): V | El mismo que el anterior. | K=25 | Retorna un arrayList de size 9 donde las posiciónes corresponder al recorrido inorden en reversa del árbol. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba 5: Verifica que el método searchLowerThan busca exitosamente los nodos del árbol que sean menores a un key dado. | | | | |
| Clase | Método | Escenario | Entrada | Resultado |
| BinaryTree | +searchLowerThan(K): V | Se ha creado un árbol binario | K=10 | El método retorna un ArrayList vacío. |
| BinaryTree | +searchLowerThan(K): V | Se ha creado un árbol binario y se la ha añadido un nodo con key= 10 y value= 1 | K=10 | El método retorna un arrayList vacío. |
| BinaryTree | +searchLowerThan(K): V | El mismo que el anterior y se le añaden los siguientes nodos:  Key=10 value=2  Key=10 value=3 | K= 10 | Retorna un arrayList vacío. |
| BinaryTree | +searchLowerThan(K): V | El mismo que el anterior y se la añade un nodo con  key=5 y value=4 | K=10 | Retorna un arraylist de size 1 donde en su posición 0 tiene el value 4. |
| BinaryTree | +searchLowerThan(K): V | El mismo que el anterior | K=6 | Retorna un arrayList de size 1 que tiene en su posición 0 el value 4 |
| BinaryTree | +searchLowerThan(K): V | El mismo que el anterior | K=4 | Retorna un arrayList vacío |
| BinaryTree | +searchLowerThan(K): V | El mismo que el anterior y se le añaden nodos  key=20 y value=5  Key=6 Value=6  Key=15 Value=7  Key=25 Value=8  Key=15 value=9 | K=10 | Retorna un arrayList de size 2 donde:  Posición 0=4  Posición 1=6 |
| BinaryTree | +searchLowerThan(K): V | El mismo que el anterior. | K=15 | Retorna un arrayList de size 5 donde:  Posición 0=1  Posición 1=2  Posición 2=3  Posición 3=4  Posición 4=6 |
| BinaryTree | +searchLowerThan(K): V | El mismo que el anterior. | K=25 | Retorna un arrayList de size 8 donde:  Posición 0=5  Posición 1=7  Posición 2=9  Posición 3=1  Posición 4=2  Posición 5=3  Posición 6=4  Posición 7=6 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba 6: Verifica que el método searchGreaterOrEqualThan busca exitosamente los nodos del árbol que sean mayores o iguales a un key dado. | | | | |
| Clase | Método | Escenario | Entrada | Resultado |
| BinaryTree | +searchGreaterOrEqualThan (K): V | Se ha creado un árbol binario | K=10 | El método retorna un ArrayList vacío. |
| BinaryTree | +searchGreaterOrEqualThan (K): V | Se ha creado un árbol binario y se la ha añadido un nodo con key= 10 y value= 1 | K=10 | El método retorna un arrayList de size 1 donde su posición 0 es 1. |
| BinaryTree | +searchGreaterOrEqualThan (K): V | El mismo que el anterior y se le añaden los siguientes nodos:  Key=10 value=2  Key=10 value=3 | K= 10 | Retorna un arrayList de size 3 donde sus posiciones son 3, 2 y 1 respectivamente. |
| BinaryTree | +searchGreaterOrEqualThan (K): V | El mismo que el anterior y se la añade un nodo con  key=5 y value=4 | K=10 | Retorna un arrayList de size 3 donde sus posiciones son 3, 2 y 1 respectivamente. |
| BinaryTree | +searchGreaterOrEqualThan (K): V | El mismo que el anterior | K=6 | Retorna un arrayList de size 3 donde sus posiciones son 3, 2 y 1 respectivamente. |
| BinaryTree | +searchGreaterOrEqualThan (K): V | El mismo que el anterior | K=4 | Retorna un arrayList de tamaño 4 donde:  Posición 0=4  Posición 1=3  Posición 2=2  Posición 3=1 |
| BinaryTree | +searchGreaterOrEqualThan (K): V | El mismo que el anterior y se le añaden nodos:  key=20 value=5  Key=6 Value=6  Key=15 Value=7  Key=25 Value=8  Key=15 value=9 | K=15 | Retorna un arrayList de size 4 donde:  Posición 0=9  Posición 1=7  Posición 2=5  Posición 3=8 |
| BinaryTree | +searchGreaterOrEqualThan (K): V | El mismo que el anterior. | K=20 | Retorna un arrayList de size 2 donde:  Posición 0=5  Posición 1=8 |
| BinaryTree | +searchGreaterOrEqualThan (K): V | El mismo que el anterior. | K=25 | Retorna un arrayList de size 1 donde su posición 0 es 8 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba 7: Verifica que el método searchGreaterThan busca exitosamente los nodos del árbol que sean mayores a un key dado. | | | | |
| Clase | Método | Escenario | Entrada | Resultado |
| BinaryTree | +searchGreaterThan(K): V | Se ha creado un árbol binario | K=10 | El método retorna un ArrayList vacío. |
| BinaryTree | +searchGreaterThan(K): V | Se ha creado un árbol binario y se la ha añadido un nodo con key= 10 y value= 1 | K=10 | Retorna un ArrayList vacío. |
| BinaryTree | +searchGreaterThan(K): V | El mismo que el anterior y se le añaden los siguientes nodos:  Key=10 value=2  Key=10 value=3 | K= 10 | Retorna un ArrayList vacío. |
| BinaryTree | +searchGreaterThan(K): V | El mismo que el anterior y se la añade un nodo con  key=5 y value=4 | K=10 | Retorna un ArrayList vacío. |
| BinaryTree | +searchGreaterThan(K): V | El mismo que el anterior | K=6 | Retorna un arrayList de size 3 donde sus posiciones son 3, 2 y 1 respectivamente. |
| BinaryTree | +searchGreaterThan(K): V | El mismo que el anterior | K=4 | Retorna un arrayList de tamaño 4 donde:  Posición 0=4  Posición 1=3  Posición 2=2  Posición 3=1 |
| BinaryTree | +searchGreaterThan(K): V | El mismo que el anterior y se le añaden nodos:  key=20 value=5  Key=6 Value=6  Key=15 Value=7  Key=25 Value=8  Key=15 value=9 | K=15 | Retorna un arrayList de size 2 donde:  Posición 0=5  Posición 1=8 |
| BinaryTree | +searchGreaterThan(K): V | El mismo que el anterior. | K=20 | Retorna un arrayList de size 1 donde su posición 0 es 8 |
| BinaryTree | +searchGreaterThan(K): V | El mismo que el anterior. | K=25 | Retorna un arrayList vacío. |

|  |
| --- |
| **TAD TreeNode** |
| ABB { value, parent, left, right } |
| inv: left.value < value  right.value > value |
| Operaciones básicas   * TreeNode value → TreeNode * isLeaf TreeNode → Boolean * getParent parent x TreeNode → parent x TreeNode * getLeft left x TreeNode → left x TreeNode * getRight left x TreeNode → right x TreeNode * setValue value x TreeNode → TreeNode * getValue TreeNode → value x TreeNode * addNode TreeNode → TreeNode * searchNode value x TreeNode → TreeNode * deleteNode value x TreeNode → TreeNode |

Operaciones

|  |
| --- |
| **Operacion**  “Descripcion”  Pre:  Post: |
|  |
|  |