|  |
| --- |
| **TAD ABB** |
| ABB { root, weight } |
| inv: |
| Operaciones básicas   * CreateABB → ABB * getRoot root x ABB → root * getWeight ABB → weight x Integer * getMax * getMin * insert Node → ABB * modify Node x ABB → ABB * search value → Node * isInTree Key → boolean * delete Node x ABB → Node |

Operaciones

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

**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 | V=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 | V=5 | El método retorna null |
| BinaryTree | +search(K): V | El mismo que el anterior | V= 10 | Retorna 1 |
| BinaryTree | +search(K): V | El mismo que el anterior y se la añade un nodo con key=5 y value=2 | V=6 | El método retorna null |
| BinaryTree | +search(K): V | El mismo que el anterior | V=5 | Retorna 2 |
| BinaryTree | +search(K): V | El mismo que el anterior y se le añade un nodo con key=20 y value 3 | V=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 | V=25 | Retorna 6, es el mismo resultado que el método getMax |
| BinaryTree | +search(K): V | El mismo que el anterior | V=5 | Retorna 2, es el mismo resultado que el método getMin |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba 3: Verifica que el método deleteNode elimina exitosamente un Nodo dado el valor a eliminar. | | | | |
| Clase | Método | Escenario | Entrada | Resultado |
| BinaryTree |  | Se ha creado un árbol binario | Value=10 | Ocurre un error  DoesntExistException |
| BinaryTree |  | Se ha creado un árbol binario y se la ha añadido un nodo con value 10 | Value=5 | Ocurre un error  DoesntExistException |
| BinaryTree |  | El mismo que el anterior | Value = 10 | El árbol está vacío. |
| BinaryTree |  | Se ha creado un árbol binario y le se ha añadido 10 y 5 en ese orden. | Value=5 | La raíz es una hoja. |
| BinaryTree |  | El mismo que el anterior | Value=10 | La raíz es el nodo 5. |
| BinaryTree |  | Se ha creado un árbol binario y le se ha añadido 10, 5, 20 en ese orden. | Value=20 | El hijo derecho de la raíz es null. |
| BinaryTree |  | Se ha creado un árbol binario y le se ha añadido 10, 5, 20, 6, 15, 25 en ese orden. | Value=20 | El hijo derecho de la raíz es 15. |
| BinaryTree |  | El mismo que el anterior | Value=5 | El hijo izquierdo de la raíz es 6. |

|  |
| --- |
| **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: |
|  |
|  |