Alejandro Fonseca Forero A00362167

DOCUMENTACIÓN PRUEBAS LABORATORIO 4 - CARRERAS

Requerimientos funcionales:

1. Generar un número aleatorio con una semilla específica. La semilla será randomizada por cada ejecución del programa.

2. Agregar un número dado a una estructura ArrayList. El número será agregado en la última posición disponible en la estructura.

3. Buscar un número dado en una estructura ArrayList de manera iterativa. La búsqueda se realiza desde el primer número de la estructura hasta el último. Se informará si el número fue encontrado o no al final de la búsqueda.

4. Buscar un número dado en una estructura ArrayList de manera recursiva. La búsqueda se realiza desde el primer número de la estructura hasta el último. Se informará si el número fue encontrado o no al final de la búsqueda.

5. Remover un número dado en una estructura ArrayList de manera iterativa. La remoción se realiza buscando desde el primer número de la estructura hasta el último, y removiendo el número que coincida con el número a remover. Se informará si el número fue removido o no al final de la remoción.

6. Remover un número dado en una estructura ArrayList de manera recursiva. La remoción se realiza buscando desde el primer número de la estructura hasta el último, y removiendo el número que coincida con el número a remover. Se informará si el número fue removido o no al final de la remoción.

7. Agregar un número dado a una lista doblemente enlazada no circular de manera iterativa. El número será agregado en la última posición disponible en la estructura.

8. Agregar un número dado a una lista doblemente enlazada no circular de manera recursiva. El número será agregado en la última posición disponible en la estructura.

9. Buscar un número dado en una lista doblemente enlazada no circular de manera iterativa. La búsqueda se realiza desde el primer número de la estructura hasta el último. Se informará si el número fue encontrado o no al final de la búsqueda.

10. Buscar un número dado en una lista doblemente enlazada no circular de manera recursiva. La búsqueda se realiza desde el primer número de la estructura hasta el último. Se informará si el número fue encontrado o no al final de la búsqueda.

11. Remover un número dado en una lista doblemente enlazada no circular de manera iterativa. La remoción se realiza buscando desde el primer número de la estructura hasta el último, y removiendo el número que coincida con el número a remover. Se informará si el número fue removido o no al final de la remoción.

12. Remover un número dado en una lista doblemente enlazada no circular de manera recursiva. La remoción se realiza buscando desde el primer número de la estructura hasta el último, y removiendo el número que coincida con el número a remover. Se informará si el número fue removido o no al final de la remoción.

13. Agregar un número dado a un árbol binario de búsqueda de manera iterativa. Si la estructura está vacía, será asignado como la raíz. Si no está vacía, se dirigirá por la rama izquierda si es menor al número padre, y por la rama derecha si es mayor. Cada vez que se agregue un número se deberá comprobar desde el primer elemento (raíz) a donde se ubicará el número.

14. Agregar un número dado a un árbol binario de búsqueda de manera recursiva. Si la estructura está vacía, será asignado como la raíz. Si no está vacía, se dirigirá por la rama izquierda si es menor al número padre, y por la rama derecha si es mayor. Cada vez que se agregue un número se deberá comprobar desde el primer elemento (raíz) a donde se ubicará el número.

15. Buscar un número dado en un árbol binario de búsqueda de manera iterativa. La búsqueda empieza con la raíz, y si el número no coincide con el número de esta, se bajará por la rama izquierda si el número es menor al de la raíz o por la rama derecha si es mayor, y se repetirá el proceso con cada elemento que se tope hasta llegar a una hoja. Se informará si el número fue encontrado o no al final de la búsqueda.

16. Buscar un número dado en un árbol binario de búsqueda de manera recursiva. La búsqueda empieza con la raíz, y si el número no coincide con el número de esta, se bajará por la rama izquierda si el número es menor al de la raíz o por la rama derecha si es mayor, y se repetirá el proceso con cada elemento que se tope hasta llegar a una hoja. Se informará si el número fue encontrado o no al final de la búsqueda.

17. Remover un número dado en un árbol binario de búsqueda de manera iterativa. La remoción empieza con la raíz, y si el número a remover no coincide con el número de esta, se bajará por la rama izquierda si el número es menor al de la raíz o por la rama derecha si es mayor, y se repetirá el proceso con cada elemento que se tope hasta llegar a una hoja. Se informará si el número fue removido o no al final de la remoción.

18. Realizar una carrera de algoritmos. Existen dos categorías (Iterativo, recursivo) y 3 modos de carrera (Agregar, buscar, eliminar), resultando en un total de 6 posibles carreras. En una carrera, el programa debe hacer el proceso que el usuario indique (Agregar iterativo, por ejemplo) en cada una de las 3 estructuras del programa: ArrayList, lista doblemente enlazada no circular, y árbol binario de búsqueda. Cada estructura deberá tener un hilo donde realizar su carrera para no interrumpir a los demás.

19. Tomar el tiempo de ejecución de cada método de los participantes. Por participante se entiende las 3 estructuras del programa. El tiempo inicia a contar cuando la carrera comienza, y cada participante se le asignará su tiempo de ejecución conforme terminen su método. El tiempo parará de contar cuando el último participante reciba su tiempo de ejecución.

20. Realizar una animación de dos círculos cambiando de tamaño. Ellos aumentarán/disminuirán de tamaño hasta cuando lleguen simultáneamente al tamaño máximo/mínimo. Una vez ahí, realizan la acción opuesta (Si aumentaba, disminuye y viceversa). Empieza cuando el tiempo empieza a contar y para cuando este para.

Requerimientos no funcionales:

1. Los requerimientos funcionales “iterativos” deben realizar sus funciones con el uso de ciclos; for o while.
2. Los requerimientos funcionales “recursivos” deben realizar sus funciones con el uso de recursividad; llamados a métodos recursivos.
3. Cada vez que se inicia una carrera, se eliminan todos los números que se hayan generado previamente en las estructuras.
4. En caso que una carrera resulte en un stackOverFlowError, el participante será descalificado de la carrera.

Casos de prueba:

Scenarios Setup

Name	**Class**	**Scenario**
setup1	TournamentTest	returns a long array = {1,2,3,4,5,6,7,8,9,10}
setup2	TournamentTest	returns a long array = {50,100,0,25,-25,125,75,30}

Test cases design

Test objective:
To check if addArrayList works properly, adding an array of known numbers and then asserting the numbers’ existence in the ArrayList.

Class	Method	Scenario	Input values	Result
Tournament	addArrayList	setup1	setup’s numbers in a cycle	True (After checking setup’s and ArrayList have the same numbers in the same positions).

Test objective:
To check if searchArrayListIterative works properly, searching the first, middle, last and a non existent number in the ArrayList.

Class	Method	Scenario	Input values	Result
Tournament	searchArrayListIterative	setup1	searchArrayListIterative(1) searchArrayListIterative(5) searchArrayListIterative(10) searchArrayListIterative(50)	True True True False

Test objective:
To check if searchArrayListRecursive works properly, searching the first, middle, last and a non existent number in the ArrayList.

Class	Method	Scenario	Input values	Result
Tournament	searchArrayListRecursive	setup1	searchArrayListRecursive(1, 0) searchArrayListRecursive(5, 0) searchArrayListRecursive(10, 0) searchArrayListRecursive(50, 0)	True True True False

Test objective:
To check if removeArrayListIterative works properly, removing and then searching the removed numbers, and removing a non existent number in the ArrayList.

Class	Method	Scenario	Input values	Result
Tournament	removeArrayListIterative	setup1	removeArrayListIterative(1) searchArrayListIterative(1) removeArrayListIterative(5) searchArrayListIterative(5) removeArrayListIterative(10) searchArrayListIterative(10) removeArrayListIterative(50)	True False True False True False False

Test objective:
To check if removeArrayListRecursive works properly, removing and then searching the removed numbers, and removing a non existent number in the ArrayList.

Class	Method	Scenario	Input values	Result
Tournament	removeArrayListRecursive	setup1	removeArrayListRecursive(1, 0) searchArrayListRecursive(1, 0) removeArrayListRecursive(5, 0) searchArrayListRecursive(5, 0) removeArrayListRecursive(10, 0) searchArrayListRecursive(10, 0)	True False True False True False

			removeArrayListRecursive(50, 0)	False

Test objective:
To check if addLinkedListIterative works properly, adding an array of known numbers and then asserting the numbers' existence in the Linked List.

Class	Method	Scenario	Input values	Result
Tournament	addLinkedListIterative	setup1	setup's numbers in a cycle	True (After checking setup's and LinkedList have the same numbers in the same positions).

Test objective:
To check if addLinkedListRecursive works properly, adding an array of known numbers and then asserting the numbers' existence in the Linked List.

Class	Method	Scenario	Input values	Result
Tournament	addLinkedListRecursive	setup1	setup's numbers in a cycle	True (After checking setup's and LinkedList have the same numbers in the same positions).

Test objective:
To check if searchLinkedListIterative works properly, searching the first, middle, last and a non existent number in the Linked List.

Class	Method	Scenario	Input values	Result
Tournament	searchLinkedListIterative	setup1	searchLinkedListIterative(1) searchLinkedListIterative(5) searchLinkedListIterative(10) searchLinkedListIterative(50)	True True True False

Test objective:
To check if searchLinkedListRecursive works properly, searching the first, middle, last and a non existent number in the Linked List.

Class	Method	Scenario	Input values	Result
Tournament	searchLinkedListRecursive	setup1	searchLinkedListRecursive(1, first) searchLinkedListRecursive(5, first) searchLinkedListRecursive(10, first) searchLinkedListRecursive(50, first) first = first linked list element	True True True False

Test objective:
To check if removeLinkedListIterative works properly, removing and then searching the removed numbers, and removing a non existent number in the Linked List.

Class	Method	Scenario	Input values	Result
Tournament	removeLinkedListIterative	setup1	removeLinkedListIterative(1) searchLinkedListIterative(1) removeLinkedListIterative(5) searchLinkedListIterative(5) removeLinkedListIterative(10) searchLinkedListIterative(10) removeLinkedListIterative(50)	True False True False True False False

Test objective:
To check if removeLinkedListRecursive works properly, removing and then searching the removed numbers, and removing a non existent number in the Linked List.

Class	Method	Scenario	Input values	Result
Tournament	removeLinkedListRecursive	setup1	removeLinkedListRecursive(1, first) searchLinkedListRecursive(1, first) removeLinkedListRecursive(5, first) searchLinkedListRecursive(5, first) removeLinkedListRecursive(10, first) searchLinkedListRecursive(10, first) removeLinkedListRecursive(50, first) first = first linked list element	True False True False True False False

Test objective:
To check if addBinaryTreeIterative works properly, adding an array of known numbers and then asserting the numbers' existence in the Binary Tree by taking the postorder traversal of the tree and matching it with the expected String.

Class	Method	Scenario	Input values	Result
Tournament	addBinaryTreeIterative	setup2 setup1	setup's numbers in a cycle setup's numbers in a cycle	True True

Test objective:
To check if addBinaryTreeRecursive works properly, adding an array of known numbers and then asserting the numbers' existence in the Binary Tree by taking the postorder traversal of the tree and matching it with the expected String.

Class	Method	Scenario	Input values	Result
Tournament	addBinaryTreeRecursive	setup2 setup1	setup's numbers in a cycle setup's numbers in a cycle	True True

Test objective:
To check if searchBinaryTreeIterative works properly, searching 3 existent numbers and a non existent number in the Binary Tree.

Class	Method	Scenario	Input values	Result
Tournament	searchBinaryTreeIterative	setup1	searchBinaryTreeIterative(1) searchBinaryTreeIterative(5) searchBinaryTreeIterative(10) searchBinaryTreeIterative(13)	True True True False

Test objective:
To check if searchBinaryTreeRecursive works properly, searching 3 existent numbers and a non existent number in the Binary Tree.

Class	Method	Scenario	Input values	Result
Tournament	searchBinaryTreeRecursive	setup1	searchBinaryTreeRecursive(1, root) searchBinaryTreeRecursive(5, root) searchBinaryTreeRecursive(10, root) searchBinaryTreeRecursive(13, root) root = binary tree root	True True True False

Test objective:
To check if removeBinaryTreeIterative works properly, removing and then searching the removed numbers, and removing a non existent number in the Binary Tree.

Class	Method	Scenario	Input values	Result
Tournament	removeBinaryTreeIterative	setup1	removeBinaryTreeIterative(1) searchBinaryTreeIterative(1) removeBinaryTreeIterative(5) searchBinaryTreeIterative(5) removeBinaryTreeIterative(10) searchBinaryTreeIterative(10) removeBinaryTreeIterative(50)	True False True False True False False

Test objective:
To check if removeBinaryTreetRecursive works properly, removing and then searching the removed numbers, and removing a non existent number in the Linked List.

Class	Method	Scenario	Input values	Result
Tournament	removeBinaryTreetRecursive	setup2	removeBinaryTreeRecursive(50, root) searchBinaryTreeRecursive(50, root) removeBinaryTreeRecursive(25, root) searchBinaryTreeRecursive(25, root) removeBinaryTreeRecursive(-25, root)	True False True False True

			searchBinaryTreeRecursive(-25, root) removeBinaryTreeRecursive(13, root) root = binary tree root	False False