UNIVERSIDAD ICESI
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE TECNOLOGIAS DE LA INFORMACIÓN

Informe de laboratorio 3:

MÉTODO DE INGENIERÍA APLICADO A LA BÚSQUEDA ESTADÍSTICA DE RENDIMIENTO DE LOS JUGADORES DE BALONCESTO PERTENECIENTES A FIBA.

Algorítmos y estructuras de datos

Profesor: Andrés Aristizabal.

Laura Hincapie
James Montealegre
Julian Mabesoy
Nicolás Taborda

2018

Fase 1: Entendimiento del problema.

Problema: EL poco aprovechamiento de todos los datos (números, cifras, marcadores, etc.) obtenidos durante un encuentro deportivo de baloncesto. Teniendo en cuenta que esta información podría llegar a ser decisiva en el entendimiento del deporte hoy en día y hacía donde este se dirige.

Requerimientos funcionales:

1. Realizar consultas de manera rápida de las estadísticas de los jugadores pertenecientes a la FIBA.
   - Resumen: Los usuarios podrán realizar consultas y obtener la información de los distintos jugadores de la FIBA por medio de, uno o varios parámetros de búsqueda.
   - Entradas: Parámetro de búsqueda a escogencia del usuario
   - Salida: Resultados estadísticos de un jugador de baloncesto perteneciente a FIBA.

2. Ingresar los datos estadísticos de los jugadores de manera manual o por medio de archivos de texto.
   - Resumen: El usuario podrá ingresar los datos estadísticos de los jugadores al sistema.
   - Entradas: Dataset o escritura de los datos de los jugadores.
   - Salida: Conjunto de datos asociados a las participaciones y encuentros de los diferentes jugadores.

3. Guardar datos de los jugadores en formato CSV.
   - Resumen: Persistir la información asociada a los jugadores.
   - Entradas: Conjunto de datos
   - Salida: Archivo en formato CSV que contiene la información de los jugadores.

Fase 2: Búsqueda de información.

- Árbol estructura de datos

Un árbol es una colección de cero o más nodos vinculados con una relación de jerarquía. Un árbol con cero nodos se denomina árbol vacío.
Un árbol tiene un nodo especial, o punto de entrada a la estructura, denominado raíz. La raíz puede tener cero o más nodos accesibles desde ella. El conjunto de esos nodos forma subárboles de la raíz, y son nodos descendientes de la raíz. La raíz es el ancestro de sus descendientes. El nodo raíz no tiene ancestros.
Un subárbol es un nodo con todos sus descendientes. Un nodo sin descendientes es una hoja. Una hoja no tiene nodos hijos. Un árbol es un: árbol vacío o un nodo simple o un nodo que tiene árboles descendientes. La definición es recursiva. Se define un árbol en términos de árboles. Una trayectoria del nodo ni al nodo nk, es una secuencia de nodos desde ni hasta nk, tal que ni es el padre de ni+1. Existe un solo enlace o vínculo entre un padre y sus hijos. Largo de una trayectoria es el número de enlaces en la trayectoria. Una trayectoria de k nodos tiene largo k-1. Alto de un nodo: largo de la trayectoria más larga de

ese nodo a una hoja. Profundidad de un nodo: es el largo de la trayectoria de la raíz a ese nodo.
El nodo con valor 3 es la raíz. Los nodos con valores: 1, 5 y 9 son hojas. Los nodos con valores: 4, 6 y 8 son nodos internos. El nodo con valor 6 es hijo de 3 y padre de 8. El nodo con valor 4 es ancestro del nodo con valor 5. {8, 9} y {4, 5} son subárboles.
El nodo con valor 1 es subárbol de la raíz. Los nodos con valores: 1, 6 y 0 son hermanos, por tener el mismo padre. El conjunto de nodos con valores: {3, 6, 4, 5} es una trayectoria, de largo 3. Alto del nodo con valor 6 es 2. La profundidad del nodo con valor 5 es 3. Todos los nodos que están a igual profundidad están en el mismo nivel. La profundidad del árbol es la profundidad de la hoja más profunda. Se dice que un árbol es una estructura ordenada, ya que los hijos de un nodo se consideran ordenados de izquierda a derecha. También es una estructura orientada, ya que hay un camino único desde un nodo hacia sus descendientes.

- Árbol binario

Cada nodo puede tener: un hijo izquierdo, o un hijo derecho, o ambos o sin hijos. A lo más cada nodo puede tener dos hijos. Un árbol binario está formado por un nodo raíz y un subárbol izquierdo I y un subárbol derecho D. Donde I y D son árboles binarios. Los subárboles se suelen representar gráficamente como triángulos.

- Árbol binario de búsqueda

Para cada nodo de un árbol binario de búsqueda debe cumplirse la propiedad: Las claves de los nodos del subárbol izquierdo deben ser menores que la clave de la raíz. Las claves de los nodos del subárbol derecho deben ser mayores que la clave de la raíz.

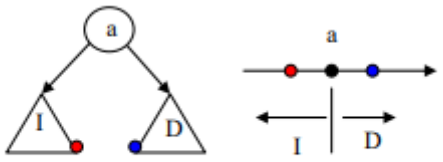


Figura 6.3. Árbol binario de búsqueda.

Esta definición no acepta elementos con claves duplicadas.
Se indican en el diagrama de la Figura 6.3, el descendiente del subárbol izquierdo con mayor clave y el descendiente del subárbol derecho con menor valor de clave; los cuales son el antecesor y sucesor de la raíz. El siguiente árbol no es binario de búsqueda, ya que el nodo con clave 2, ubicado en el subárbol derecho de la raíz, tiene clave menor que ésta.

Tomado de: www2.elo.utfsm.cl/~lsb/elo320/clases/c6.pdf

- Árbol AVL

Un árbol AVL es un árbol binario de búsqueda que cumple con la condición de que la diferencia entre las alturas de los subárboles de cada uno de sus nodos es, como mucho 1. La denominación de árbol AVL viene dada por los creadores de tal estructura (Adelson-Velskii y Landis). Recordamos que un árbol binario de búsqueda es un árbol binario en el cual cada nodo cumple con que todos los nodos de su subárbol izquierdo son menores que la raíz y todos los nodos del subárbol derecho son mayores que la raíz. Recordamos también que el tiempo de las

operaciones sobre un árbol binario de búsqueda son O(log n) promedio, pero el peor caso es O(n), donde n es el número de elementos. La propiedad de equilibrio que debe cumplir un árbol para ser AVL asegura que la profundidad del árbol sea O(log(n)), por lo que las operaciones sobre estas estructuras no deberán recorrer mucho para hallar el elemento deseado. Como se verá, el tiempo de ejecución de las operaciones sobre estos árboles es, a lo sumo O(log(n)) en el peor caso, donde n es la cantidad de elementos del árbol. Sin embargo, y como era de esperarse, esta misma propiedad de equilibrio de los árboles AVL implica una dificultad a la hora de insertar o eliminar elementos: estas operaciones pueden no conservar dicha propiedad.

- Árbol Rojinegro

• Árbol binario estricto (los nodos nulos se tienen en cuenta en la definición de las operaciones ⇒ todo nodo hoja es nulo)
• Cada nodo tiene estado rojo o negro
• Nodos hoja (nulos) son negros
• La raíz es negra (esta condición se impone para simplificar algunas operaciones) Se cumplen las condiciones:
1. Un nodo rojo tiene dos hijos negros.
2. Todo camino de la raíz a cualquier hoja pasa por el mismo número de nodos negros.

Tomado de: https://www.infor.uva.es/~cvaca/asigs/doceda/rojonegro.pdf

Fase 3: Búsqueda de soluciones creativas.

1. **Optimización de recursos:** La cantidad de datos a almacenar es arbitrariamente grande, razón que obliga a los programadores a buscar estrategias que permitan manejar en cada árbol NO se almacenará el índice con el valor porque la memoria se satura debido a la cantidad de datos, se almacena la clave y el valor es un String que lleva a la ruta del archivo de texto por cada registro.

2. **Implementación de los árboles desde el nodo hasta la clase principal de árbol:** Esta estrategia resulta ser una solución creativa gracias al aporte en conocimiento y experiencia que se genera sobre el desarrollador, si bien, no hay una diferencia marcada entre la implementación de todos los métodos sobre la clase principal del árbol, en el aprendizaje ofrecido por el curso inmediatamente anterior no se analizó como alternativa y por ende ofrece ventajas en la experiencia de los integrantes del grupo.

3. **Implementación priorizando los árboles rojinegros frente a los AVL:** Debido a que los árboles AVL tienen la propiedad de que su balanceo, es más estricto podríamos obtener una búsqueda más lenta y por ende para la solución del problema, ante los tres índices de búsqueda, se implementarán dos árboles rojinegros y un AVL.

4. **Utilización de ArrayList bajo los árboles binarios para obtener la información de forma directa:** dado que los árboles binarios son estructuras dinámicas, el acceso a la información se debe realizar de forma indirecta, es decir, implementar clases que permitan obtener la estructura del árbol y luego adaptar esas implementaciones al mundo del problema, solución que resulta ser un poco

tediosas. Es por esto que el grupo de desarrollo decidió almacenar cada nodo del árbol en un ArrayList para que el acceso a la información sea más fácil.

5. **Implementaciones de los métodos desde ABB:** no se implementa toda la lógica del programa en el nodo, sino que se implementará desde ABB para que ARN y AVL puedan heredar desde esta.

Fase 4: Transición de la formulación de ideas a los diseños preliminaries

De la fase anterior se deciden descartar algunas opciones, las razones se explican a continuación:

- Lo expuesto en la idea número tres en realidad es un argumento que resulta ser muy relativo, es decir, ambos tipos de árboles tienen la misma complejidad temporal para su peor o mejor caso y esta presenta variaciones según la estructura del árbol, por ende resulta irrelevante considerar cuántos árboles rojinegros o AVL se van a implementar para obtener un búsqueda más rápida.

- La afirmación de la idea número cinco es completamente cierta, acceder a la información del programa cuando las estructuras de datos que se manejan son árboles se debe de hacer de forma indirecta, pero la idea plantea poner bajo estos árboles un ArrayList, acción que saldría bastante costosa porque se presentaran aumentos en la complejidad temporal y espacial.

Ya habiendo eliminado las ideas que no resultan muy convenientes para el desarrollo del software propuesto tendremos que: Se implementarán árboles rojinegros y AVL como estrategia para realizar operaciones básicas en una complejidad temporal conveniente. También se realizarán las operaciones desde la clase del nodo y en las clases principales de los árboles solo se llamaran a estos métodos, esto como método para aumentar la experiencia del grupo de programadores. Por último se decidió almacenar la clave y valor de cada nodo en un String que me lleva a una ruta de texto como necesidad de optimizar recursos.

Fase 5: Evaluación de la mejor solución

	CRITERIO	5	4	3	2	1
C1: 20%	complejidad temporal	O(1)	O(log n)	O(n)	O(n log n)	> O(n log n)
C2: 20%	complejidad espacial	O(1)	O(log n)	O(n)	O(n log n)	> O(n log n)
C3: 30%	¿que tanto se aprende?	La combinación de estructuras de datos permite enriquecer al 100% el conocimiento sobre manejo de algoritmos.	La implementación de la estructura de datos hace un muy buen aporte.	La implementación de la estructura de datos hace un aporte medio.	La implementación de la estructura de datos no hace ningún aporte.	La implementación de esa estructura no aporta absolutamente nada al conocimiento del programador y además genera confusiones..
C4: 15%	facilidad de implementar	Implementar esta estructura no genera ninguna dificultad.	Implementar esta estructura genera dificultad baja.	Implementar esta estructura genera dificultad media.	Implementar esta estructura genera dificultad alta.	Implementar esta estructura resulta imposible para el programador.

C5: 15%	Restricción de capacidad	NO tiene restricciones de capacidad.				La capacidad es limitada.

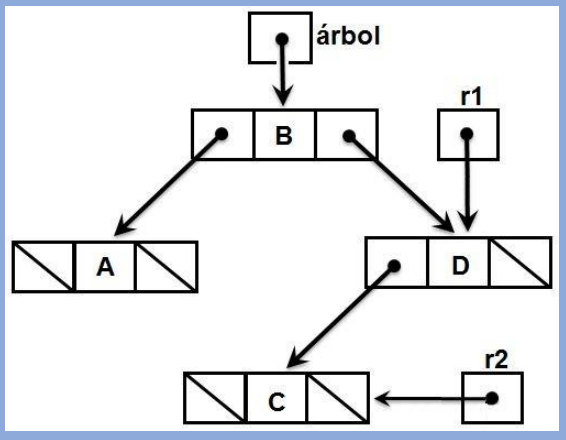
Valoración de las estructuras para la implementación de la base de datos						
Estructura	**C1**	**C2**	**C3**	**C4**	**C5**	**Total**
ABB	4	3	3	5	5	**20**
ARN	4	3	4	2	5	**17**
AVL	4	3	5	1	5	**18**

Fase 6: Elaboración de informes y especificaciones

AD ABB

TAD ABB

<K extends Comparable, V>



Invariante

Sea x un nodo del árbol. Si y es un nodo en el subarbol izquierdo de x, entonces key[y] key[x]. Si y es un nodo en el subarbol derecho de x, entonces key[y] key[x]

Operaciones primitivas

- insertar ABB → ABB
- eliminar ABB → ABB
- rotarIzquierda ABB → ABB
- rotarDerecha ABB → ABB
- mínimo ABB → Nodo
- máximo ABB → Nodo
- sucesor ABB → Nodo
- transplant ABB → ABB

Insertar
“Agregar nuevo nodo al árbol binario de búsqueda garantizando la propiedad de orden”
{pre: }
{post: se agrega el nuevo nodo al árbol}

Eliminar
“Elimina el nodo cuya clave es indicada y garantiza la propiedad de orden en el árbol”
{pre: El árbol debe de estar creado y al menos su nodo-raíz debe existir}
{post: se elimina el nodo indicado}

RotarIzquierda
“rotar hacía la izquierda el árbol cambiando su estructura sin alterar su orden”
{pre: el nodo que se desea rotar debe tener hijo izquierdo}
{post: se rota el árbol hacía la izquierda y éste conserva su orden}

RotarDerecha
“rotar hacía la derecha el árbol cambiando su estructura sin alterar su orden”
{pre: el nodo que se desea rotar debe tener hijo derecho}
{post: se rota el árbol hacía la derecha y éste conserva su orden}

Mínimo
“mínimo valor del ABB”
{pre: existe al menos un nodo en el árbol}
{post: se obtiene el valor mínimo del árbol}

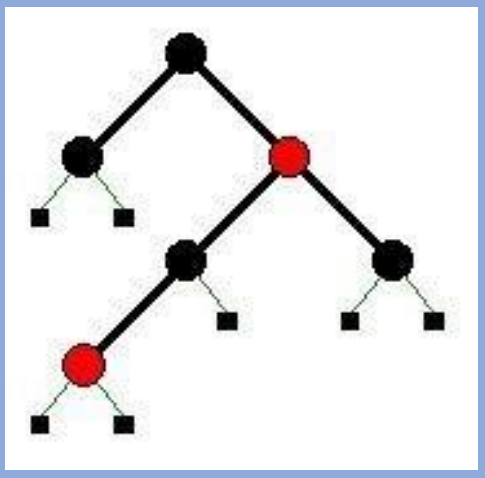
Máximo
“máximo valor del ABB”
{pre: existe por lo menos un nodo}
{post: se obtiene el valor máximo del árbol}

Sucesor
“Da el sucesor de un nodo del árbol (Dado un nodo ‘x’ donde key[x]=k, el sucesor de ‘x’ es el nodo ‘y’ tal que key[y] es la llave más pequeña, mayor que key[x])”
{pre: el árbol está creado y para encontrar el sucesor deben existir como mínimo dos nodos que cumplen con la relación descrita en la restricción}
{post: se obtiene el sucesor del elemento señalado}

TAD Rojinegro

TAD Árbol Rojo-Negro

<K extends Comparable, V>



Invariante

- Es un ABB
- Cada nodo tiene un campo extra que almacena su color
- La raíz es negra
- Todo nodo es rojo y negro

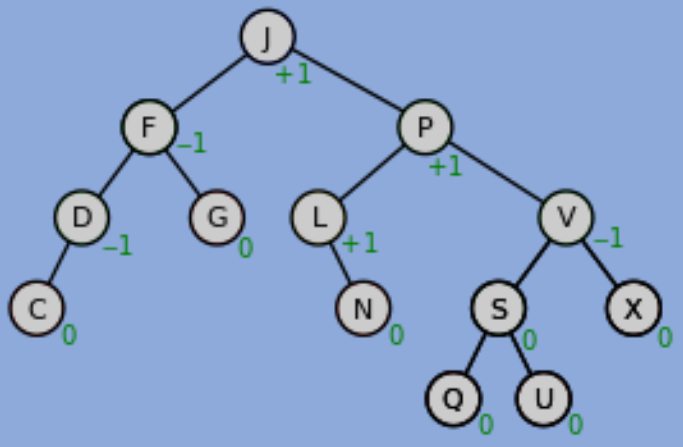
Operaciones primitivas

- insertar ARN → ARN
- eliminar ARN → ARN
- rotarIzquierda ARN → ARN
- rotarDerecha ARN → ARN
- mínimo ARN → Nodo
- máximo ARN → Nodo
- sucesor ARN → Nodo
- transplant ARN → ARN

TAD AVL

TAD AVL

<K extends Comparable, V>



Invariante

- Es un ABB
- Cada nodo cuenta con un factor de balanceo
  balanceFactor = height(left subtree) - height(right subtree)

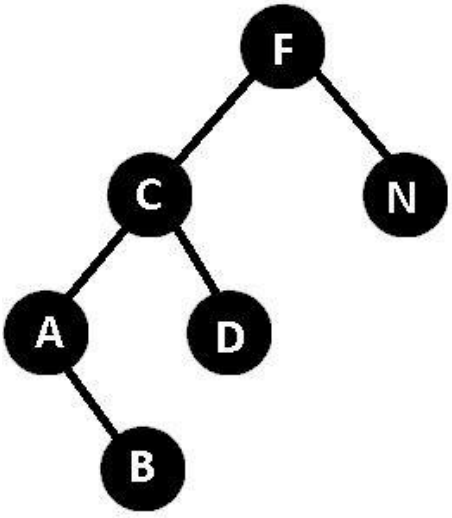
Operaciones primitivas

- insertar AVL → AVL
- eliminar AVL → AVL
- rotarIzquierda AVL → AVL
- rotarDerecha AVL → AVL
- mínimo AVL → Nodo
- máximo AVL → Nodo
- sucesor AVL → Nodo
- transplant AVL → AVL

Casos de prueba:
Casos de prueba clase ABB
Escenario 1:
Se crea un árbol binario de búsqueda con 6 nodos con las siguientes claves asociadas a un valor:

- Nodo1 con clave “F” y valor “1”
- Nodo2 con clave “C” y valor “2”
- Nodo3 con clave “N” y valor “3”
- Nodo4 con clave “A” y valor “4”
- Nodo5 con clave “B” y valor “5”
- Nodo6 con clave “D” y valor “6”

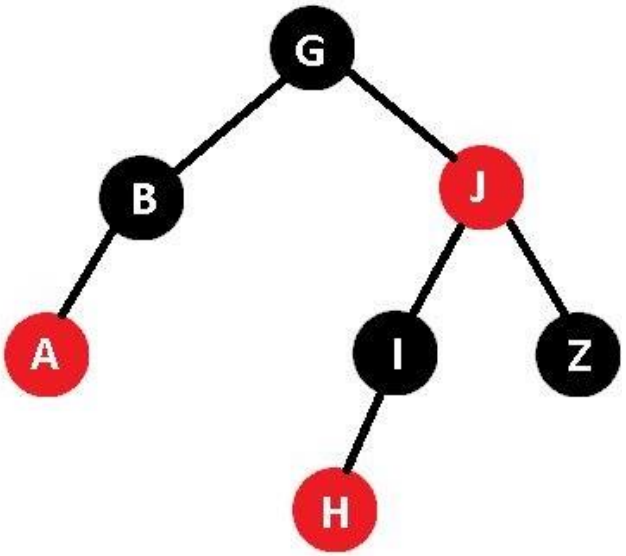
El árbol queda distribuido de la siguiente manera:



Escenario 2:
Se crea un árbol rojinegro con 7 nodos con las siguientes claves asociadas a un valor

- Nodo1 con clave “B” y valor “1”
- Nodo2 con clave “G” y valor “2”
- Nodo3 con clave “Z” y valor “3”
- Nodo4 con clave “A” y valor “4”
- Nodo5 con clave “I” y valor “5”
- Nodo6 con clave “J” y valor “6”
- Nodo7 con clave “H” y valor “6”

El árbol queda distribuido de la siguiente manera:

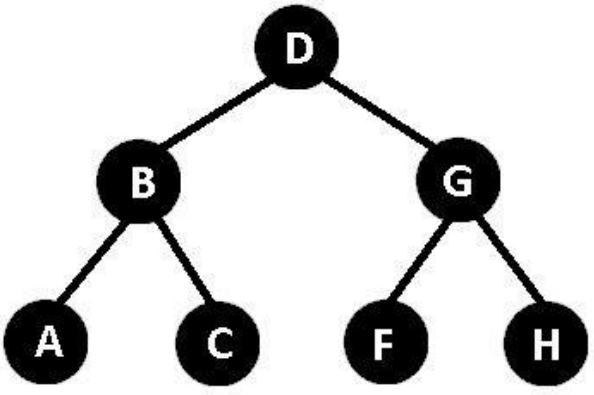


Escenario 3:

Se crea un arbol AVL con 7 nodos mostrados a continuación:

- Nodo1 con clave “A” y valor “1”
- Nodo2 con clave “B” y valor “2”
- Nodo3 con clave “C” y valor “3”
- Nodo4 con clave “D” y valor “4”
- Nodo5 con clave “F” y valor “5”
- Nodo6 con clave “G” y valor “6”
- Nodo7 con clave “H” y valor “7”

El árbol queda distribuido de la siguiente manera:



Pruebas

Objetivo prueba 1: Agregar un conjunto de datos y observar que se respeta la propiedad de orden

Clase	ABB
Método	Insertar(K k, V v)
Escenario	Escenario1
Valores de entrada	Se inserta un nuevo nodo con clave “H” y valor “7”
Resultado	El nodo se agregó según la jerarquía de orden

Objetivo prueba 2: Agregar al árbol dos nuevos registros con claves existentes y corroborar que se agregan a lista de claves correspondientes a la clave de ese nodo

Clase	ABB
Método	Insertar(K k, V v)
Escenario	Escenario1
Valores de entrada	Se inserta dos nuevos registros: Nodo(“N”, “7”) Nodo(“N”, “8”)
Resultado	Los registros se agregan y el tamaño de la lista de valores de ese nodo aumenta en una cantidad correspondiente a 3

Objetivo prueba 3: Buscar un nodo del árbol por su clave, corroborar que existe y que sus hijos sean los correspondientes a ese nodo

Clase	ABB
Método	busquedaIterativa(nodoArbol)
Escenario	Escenario1

Valores de entrada	Clave a buscar “C”
Resultado	Nodo con la clave pasada por parámetro

Objetivo prueba 4: Buscar un nodo del árbol por su clave, que contenga más de un valor en su lista de valores

Clase	ABB
Método	busquedaIterativa(nodoArbol)
Escenario	Escenario1
Valores de entrada	Se insertan dos nuevos registros: Nodo(“D”, “7”) Nodo(“D”, “8”) La búsqueda se realiza por la clave “D”
Resultado	Nodo buscado con un tamaño de 3 en su lista de valores

Objetivo prueba 5: Eliminar un nodo del árbol el cual sea una hoja y observar que el elemento se elimino conservando la propiedad de orden del árbol.

Clase	ABB
Método	Eliminar(nodoArbol)
Escenario	Escenario1
Valores de entrada	Nodo con clave “D”
Resultado	Se elimino el nodo del árbol y se conservo la propiedad de orden

Objetivo prueba 6: Eliminar un nodo del árbol el cual tenga solo un hijo y verificar que se siga conservando la propiedad de orden

Clase	ABB

Método	Eliminar(nodoArbol)
Escenario	Escenario1
Valores de entrada	Nodo con clave “A”
Resultado	Se elimino el nodo del árbol y se conservo la propiedad de orden

Objetivo prueba 7: Eliminar un nodo del árbol el cual tenga hijo izquierdo e hijo derecho y verificar que se siga conservando la propiedad de orden

Clase	ABB
Método	Eliminar(nodoArbol)
Escenario	Escenario1
Valores de entrada	Se añade un nuevo nodo al árbol con clave “H” y valor “7” Se va a eliminar el nodo con clave “F” que es la raíz del árbol
Resultado	Se elimino el nodo del árbol y se conservó la propiedad de orden

Objetivo prueba 8: Rotar un nodo a la izquierda y comprobar si la acción se realiza correctamente

Clase	ABB
Método	rotarIzquierda(nodoArbol)
Escenario	Escenario1
Valores de entrada	Nodo a rotar “A”
Resultado	Se intercambia la posición de el nodo con otro

Objetivo prueba 9: Rotar un nodo a la derecha y comprobar si la acción se realiza correctamente

Clase	ABB
Método	rotarIzquierda(nodoArbol)
Escenario	Escenario1
Valores de entrada	Se agrego el nodo con clave “H” y valor “7” Nodo a rotar “N”
Resultado	Se intercambia la posición de el nodo con otro

Objetivo prueba 10: Comprobar si el método minimo retorna el nodo con clave menor de un nodo dado

Clase	ABB
Método	minimo(nodoArbol)
Escenario	Escenario1
Valores de entrada	Nodo raíz con clave “F”
Resultado	Rotorna el nodo con clave “A”

Objetivo prueba 11: Comprobar si el método máximo retorna el nodo con clave mayor de un nodo dado

Clase	ABB
Método	maximo(nodoArbol)
Escenario	Escenario1
Valores de entrada	Nodo raíz con clave “F”
Resultado	Rotorna el nodo con clave “N”

Objetivo prueba 12: Comprobar si el método sucesor retorna el nodo sucesor de un nodo pasado por parametro

Clase	ABB

Método	sucesor(nodoArbol)
Escenario	Escenario1
Valores de entrada	Se inserta un nodo con clave “H” y valor “7” Se busca el sucesor del nodo raíz con clave “F”
Resultado	Retorna el nodo con clave “H”

Objetivo prueba 13: Insertar un nodo al árbol rojinegro y observar que se agregue, respetando la propiedad de orden y las propiedades que los arboles rojos y negros deben conservar.

Clase	ARB
Método	insertarARB(K, V)
Escenario	Escenario2
Valores de entrada	Se inserta un nuevo nodo con clave “C” y valor “7”
Resultado	Se agrega el nodo y se conservan las propiedades de los arboles R&B y la propiedad de orden

Objetivo prueba 14: Insertar nodos con claves repetidas al árbol rojinegro, observar que se agreguen los valores a la lista de claves correspondiente y verificar que se agregue, respetando la propiedad de orden y las propiedades que los arboles rojos y negros deben conservar.

Clase	ARB
Método	insertarARB(K, V)
Escenario	Escenario2

Valores de entrada	Se insertan los siguientes nodos nodoArbol(“C”, “7”) nodoArbol(“C”, “8”) nodoArbol(“C”, “9”) nodoArbol(“C”, “10”)
Resultado	Se agregan los nodos y el tamaño de la lista de valores de esa clave correspondiente es 4

Objetivo prueba 15: Eliminar un nodo en el caso mas simple, que es una hoja de color rojo, la cual al ser eliminada no se tiene que hacer modificaciones al arbol, comprobar que efectivamente el arbol solo elimina el nodo y no cambia el color ni la forma del arbol R&B

Clase	ARB
Método	eliminar (nodoArbol)
Escenario	Escenario2
Valores de entrada	Se inserta el siguiente nodo nodoArbol(“C”, “7”) Nodo por eliminar: nodo de clave “A”
Resultado	El nodo se elimina del árbol y la estructura del árbol se conserva

Objetivo prueba 16: Eliminar un nodo que tiene un único hijo, pero su eliminación implica restructurar la forma del árbol y los colores de algunos nodos. Observar que se sigan conservando todas las propiedades

Clase	ARB
Método	eliminar (nodoArbol)
Escenario	Escenario2

Valores de entrada	Se inserta el siguiente nodo nodoArbol(“C”, “7”) Nodo por eliminar: nodo de clave “I”
Resultado	El nodo se elimina del árbol, la estructura del árbol cambia para conservar las propiedades de los ARB

Objetivo prueba 17: Eliminar un nodo sin hijos, pero de color negro, Observar que las propiedades de los arboles R&B se sigan conservando

Clase	ARB
Método	eliminar (nodoArbol)
Escenario	Escenario2
Valores de entrada	Se inserta el siguiente nodo nodoArbol(“C”, “7”) Nodo por eliminar: nodo de clave “Z”
Resultado	El nodo se elimina del árbol, la estructura del árbol cambia para conservar las propiedades de los ARB

Objetivo prueba 18: Insertar un árbol AVL el cual no genere una alteración en la estructura del árbol

Clase	ARB
Método	insertar (K, V)
Escenario	Escenario3
Valores de entrada	Se inserta el siguiente nodo nodoArbol(“I”, “8”)
Resultado	El nodo se Inserta y se conserva las propiedades de los arboles AVL

Objetivo prueba 19: Eliminar un nodo de árbol AVL en el caso mas básico, donde solo es una hoja y su eliminación no involucra balanceos

Clase	ARB
Método	eliminar (nodoArbol)
Escenario	Escenario3
Valores de entrada	Se inserta el siguiente nodo nodoArbol(“A”, “1”)
Resultado	El nodo se elimino y el árbol conserva aun su estructura

Objetivo prueba 20: Eliminar dos nodos de árbol AVL en el cual se tendrá que cambiar el factor de balanceo de algunos nodos

Clase	ARB
Método	eliminar (nodoArbol)
Escenario	Escenario3
Valores de entrada	Se inserta el siguiente nodo nodoArbol(“A”, “1”) nodoArbol(“B”, “2”)
Resultado	El nodo se elimino y el los nodos adquirieron nuevos factores de balanceo correctos