# MÉTODO DE LA INGENIERÍA

## 1. *Identificación del problema:*

### Contexto del problema:

La Bolsa de Valores de Colombia (BVC) es una bolsa multi-producto y multi-mercado que administra los sistemas de negociación y registro de los mercados de acciones, renta fija, derivados, divisas, OTC y servicios de emisores en Colombia. Actualmente la BVC no permite transar con acciones internacionales ni trabajar con el mercado de divisas o de derivados.

### Definición del problema:

La BVC ha aprovechado esta coyuntura y con la ayuda del gobierno nacional quiere consolidar en una aplicación, los datos de algunos mercados de divisas y de acciones internacionales. Se espera que la aplicación permite realizar análisis sobre estos datos, se conozcan patrones acerca de los movimientos de los mercados, qué criterios toman más fuerza.

Se espera que se puedan hacer las siguientes operaciones:

- Consultar el precio más alto de una acción o un mercado de divisas en un rango de tiempo. Es decir, el precio más alto de la misma, dada una fecha inicial y una fecha final.
- Consultar el precio más bajo de una acción o un mercado de divisas en un rango de tiempo. Es decir, el precio más alto de la misma dada una fecha inicial y una fecha final.
- Consultar el periodo de tiempo donde una acción / mercado de divisas tuvo su mayor crecimiento.
- Mostrar una gráfica del estado de los precios de una acción / mercado de divisas. En la gráfica debe ser posible agregar hasta un máximo de 3 acciones / mercados de divisas en donde cada indicador de gráfica deberá tener un color diferente.
- Cuáles acciones / Mercado de divisas superan un valor en un rango de tiempo.
- Cuales son las 3 acciones / Mercados que presentaron mayor crecimiento en un rango de tiempo.

### Identificación de necesidades y síntomas:

- La solución a los problemas planteados anteriormente debe ser lo más eficiente posible.

## 2. ***Recopilación de información:***

**Mercado financiero:**
Es un espacio (físico o virtual o ambos) en el que se realizan los intercambios de instrumentos financieros y se definen sus precios. En general, cualquier mercado de materias primas podría ser considerado como un mercado financiero si el propósito del comprador no es el consumo inmediato del producto, sino el retraso del consumo en el tiempo.

**Acciones financieras:**
Cuando hablamos de acciones hablamos de un título que emite una sociedad. Este título es una fracción del valor del capital social de la empresa. Todas las acciones de una misma sociedad son fracciones idénticas del capital social.
Los individuos que poseen dichas acciones son conocidos como accionistas. Cuantas más acciones se posean de una empresa, más porcentaje de la empresa recae en el accionista. Esto pone al accionista en una situación de poder, ya que puede tomar decisiones y exigir sus derechos sobre la empresa. Tampoco hay que olvidar que el accionista ha de cumplir también con sus obligaciones.

**Tipos de acciones financieras:**
Las acciones se pueden dividir en diferentes tipos. Los más habituales son sin duda los siguientes:

- **Acciones financieras comunes u ordinarias.** Estas son las acciones más habituales. Ofrecen al propietario el derecho a voto en la Junta de Accionistas y a recibir parte de los beneficios de la empresa anualmente. Los beneficios (o dividendos que reciben) se obtienen tras abonar los intereses a los obligacionistas y a las acciones preferentes.
- **Acciones financieras preferentes.** Las acciones preferentes dan más poder a los accionistas que las de tipo ordinario. Por ejemplo, de cara a cobrar los dividendos de la empresa, antes de pagar a los accionistas ordinarios se paga a los propietarios de acciones preferentes. La tasa de los dividendos de estas acciones se ha de fijar en el momento en el que se emiten. Y pueden ser tanto fijas como variables.
- **Acciones financieras sin voto.** Estas acciones eliminan el derecho a voto en la Junta de Accionistas. Sin embargo, aquellos que las poseen sí tienen derechos a nivel económico. ¿Qué significa esto? Que cobran los dividendos correspondientes a sus acciones.

**Capital:**
Conjunto de recursos, bienes y valores disponibles para satisfacer una necesidad o llevar a cabo una actividad definida y generar un beneficio económico o ganancia particular, está estrechamente relacionado con el comportamiento de las personas que intervienen en este aspecto.

**Divisa:**
Es la moneda extranjera respecto a un país de referencia.

**Data set:**
Un conjunto de datos corresponde a los contenidos de una única tabla de base de datos o una única matriz de datos estadística, donde cada columna de la tabla representa una variable en particular, y cada fila representa un miembro determinado del conjunto de datos en cuestión.

Para el desarrollo de este programa se nos han facilitado la información de los siguientes mercados financieros:

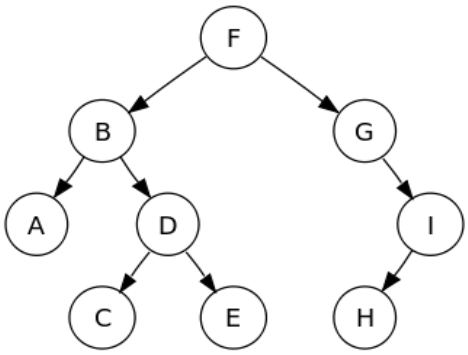
- **#US30:** es un instrumento financiero popular que se basa en el desempeño del índice Dow Jones Industrial Average Future.
- **#USSPX500:** es un índice ponderado de capitalización de mercado que incluye a 500 empresas estadounidenses de diferentes sectores. Las empresas representan más del 70% de la capitalización de mercado total del mercado de valores de EE. UU.
- **BTCUSD:** es el mercado de divisas del bitcoin.
- **EURUSD:** abreviatura para el par de divisas o cruce del **euro** y el **dólar estadounidense**, las monedas de la Zona Euro (EUR) y Estados Unidos (USD). Este par de divisas muestra cual es la cantidad de dólares estadounidenses (la **divisa de cotización**) que se requiere para comprar un euro.
- **GBPCD:** muestra cuál es la cantidad de libras esterlinas se requieren para comprar un dólar.
- **USDJPY:** es el segundo par de divisas más negociado en FOREX, el cual es formado por el dólar norteamericano y por el yen japonés.
- **WTI:** (West Texas Intermidiate) es el petróleo crudo que se extrae en el golfo de México y sirve como referencia para las transacciones financieras en New York (NYMEX).
- **XUUSD:** el par XAU/USD le dice al comerciante cuántos dólares estadounidenses (la moneda de cotización) se necesitan para comprar una onza de oro (la moneda base). En el mercado Forex, el oro es una forma de moneda. La particularidad del oro es que solo puede negociarse

con dólares de los Estados Unidos (USD). El código de oro aceptado internacionalmente es XAU. Conocido por ser un activo "seguro", se espera que aumente su valor en tiempos de volatilidad e incertidumbre económica.

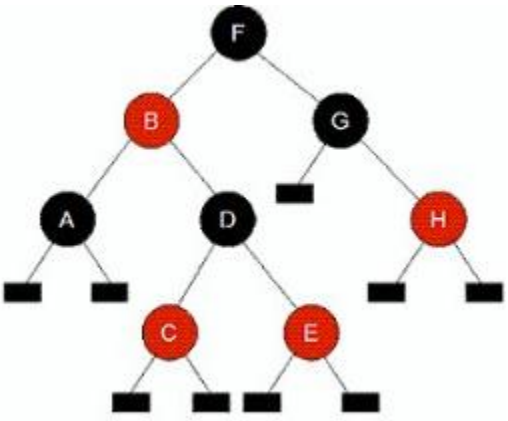
## 3. ***Búsqueda de soluciones creativas:***

Para la solución de los problemas que se han planteado anteriormente se tienen las siguientes alternativas:

**Alternativa 1: *Árboles binarios de búsqueda o Binary search trees***



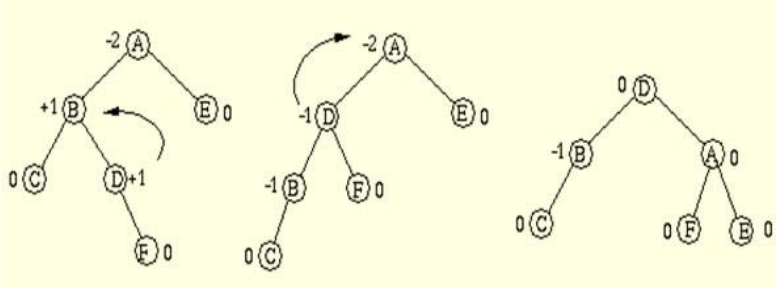

Un árbol binario es una estructura de datos en la cual cada nodo puede tener un hijo izquierdo y un hijo derecho. No pueden tener más de dos hijos (de ahí el nombre "binario"). Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo. En el caso contrario el hijo es llamado un nodo interno.




**Alternativa 2: *Árboles rojinegros o Red black trees***

Un árbol rojo negro es un tipo abstracto de datos, concretamente es un árbol binario de búsqueda equilibrado, una estructura de datos utilizada en informática y ciencias de la computación para organizar información compuesta por datos comparables. Es complejo, pero tiene un buen peor caso de tiempo de ejecución para sus operaciones y es eficiente en la práctica. Puede buscar, insertar y borrar en un tiempo O(log n), donde n es el número de elementos del árbol. En los árboles rojo-negro las hojas no son relevantes y no contienen datos. La idea sería guardar los mercados financieros con sus características y el orden de este estaría dado por su crecimiento respecto a una fecha.

**Alternativa 3: *Árboles AVL o AVL trees***



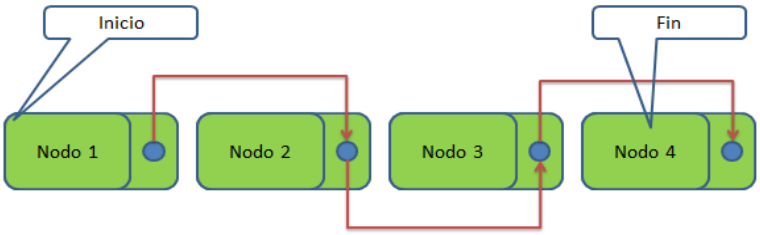
Es un tipo especial de árbol binario ideado por los matemáticos rusos Adelson-Velskii y Landis. Fue el primer árbol de búsqueda binario auto balanceado que se ideó. Los árboles AVL están siempre equilibrados de tal modo que, para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha. Gracias a esta forma de equilibrio (o balanceo), la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad O (log n). El factor de equilibrio puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles.

**Alternativa 4: *Hash table con árboles auto balanceables***

Se propone guardar cada uno de los mercados financieros en una tabla hash que contiene árboles rojinegros o avl, en los que se almacenarán los mercados por alguno de sus atributos o por estadísticas.

**Alternativa 5: *Lista enlazada o LinkedList***

Se define lista enlazada como una colección de elementos que están enlazados entre sí y que cada nodo contiene un valor.




Es la variante más simple que existe pues en esta estructura de datos tenemos un conjunto de nodos que están enlazados solo con el nodo siguiente de tal forma que si queremos recorrer la colección lo haremos del primero hasta el último, pero no se puede regresar. El programa implementaría una lista enlazada que guardará todos los mercados financieros y se ordenará basándose en su nombre, sin embargo, al momento de querer realizar alguna operación como búsqueda u otra, se hará uso del método de ordenamiento merge sort, lo cual facilitará la tarea a realizar.

**Alternativa 6: *Montículo o Heap***

Un árbol binario completo que almacena elementos con campo clave y donde los nodos cumplen la propiedad de montículo: Todo nodo del árbol almacena un elemento cuya clave es menor que las claves de sus descendientes en el árbol, esa clave también puede ser mayor que las claves de sus descendientes, todo depende de lo que se necesite en el momento. Para la solución de este problema, se propone un heap que funcione de manera dinámica, el cual almacenará cada uno de los mercados financieros y que, dependiendo de la operación a realizar, este cambie su orden para lograr las operaciones básicas en el mejor de los casos en *O* (1).

## *4. Transición de las ideas a diseños preliminares:*

Luego de hacer una profunda investigación de las posibles estructuras de datos que podrían darles solución a los problemas ya propuestos, es momento de analizar cuáles de estas resultarían más eficientes. Al hacer diferentes verificaciones se logró evidenciar que:

**Alternativa 1: *Árboles binarios de búsqueda***

- En el peor de los casos, el tiempo de ejecución de las operaciones básicas tarda O (n)

**Alternativa 2: *Árboles rojinegros***

- En el peor de los casos el tiempo de ejecución de las operaciones básicas tarda *O* (log n).
- Cuando el árbol tiene mucho peso supera la memoria principal.
- Solo puede ser consultada por un tipo de orden.

**Alternativa 4: *Hash table con árboles auto balanceables***

- En el peor de los casos el tiempo de ejecución de las operaciones básicas tarda *O* (log n).
- *Usa al menos dos estructuras de datos.*

**Alternativa 5: *Listas enlazadas***

- El tiempo de ejecución de las operaciones básicas puede llegar a tardar (n^2 * log n) debido al uso del merge sort.
- Cuando la cantidad de mercados financieros sea mucha, puede superar la memoria principal.

5. ***Evaluación y selección de la mejor solución:***

Después de analizar los aspectos necesarios para la solución de los problemas encontrados, se plantean los siguientes criterios:

***Criterio A:*** Complejidad temporal de los métodos para resolver las operaciones básicas de la solución.

- [4] Constante
- [3] Logarítmica
- [2] Lineal
- [1] Polinómica
- [0] Desconocida

***Criterio B:*** Nivel de dificultad de programación.

- [5] Muy fácil: Programación de la solución en un día.
- [4] Fácil: Programación de la solución de dos a tres días.
- [3] Regular: Programación de la solución entre diez a quince días.
- [2] Difícil: Programación de la solución entre uno y dos meses.
- [1] Muy difícil: Programación de la solución mayor a tres meses.

***Criterio C:*** Cumplimiento de los requerimientos solicitados.

- [2] Cumple con todos los requerimientos solicitados
- [1] Cumple con algunos de los requerimientos solicitados.
- [0] No cumple con ninguno de los requerimientos solicitados.

***Criterio D:*** Uso de estructuras de datos propias.

- [3] Todas las estructuras utilizas para la solución son propias.
- [2] De las tres estructuras utilizadas, 2 son propias.
- [1] De las tres estructuras utilizadas, 1 es propia.
- [0] Ninguna estructura de datos es propia.

***Criterio E:*** Cantidad de memoria principal requerida.

- [3] Se hace uso de 50mb-100mb de memoria principal.
- [2] Se hace uso de 101-250mb de memoria principal.
- [1] Se hace uso 251mb-612mb de memoria principal.
- [0] Se hace uso de 613mb-1024mb de memoria principal.

| | Criterio A | Criterio B | Criterio C | Criterio D | Criterio E | Total |
|---|---|---|---|---|---|---|
| **Alternativa 1** | 1 | 1 | 1 | 2 | 1 | 6 |
| **Alternativa 2** | 3 | 4 | 1 | 3 | 1 | 12 |
| **Alternativa 4** | 3 | 3 | 2 | 2 | 3 | 13 |

Basado en la anterior tabla, se puede inferir que la alternativa más óptima para la solución de este problema es la número 4:

- *Hash table con árboles auto balanceables*

Los criterios que se plantearon fueron de acuerdo a la solución del problema que se tenía. Por un lado, las tablas hash ayudan a la inserción y búsqueda casi inmediata por medio de llaves que son representadas por el nombre de cada uno de los mercados financieros. Por otro lado, el tipo de árbol seleccionado es el rojinegro, dado que, los árboles AVL no funcionan de la mejor manera cuando la cantidad de datos es grande.

## 6. ***Documentación:***

### ***Especificación de requerimientos:***

| **Requerimiento funcional 1.** | |
|---|---|
| **Nombre** | Consultar el precio más alto de una acción |
| **Resumen** | El programa permitirá consultar el precio más alto de un mercado de divisas en un rango de tiempo. |
| **Entrada** | |
| Nombre de la acción a consultar | |
| **Resultado** | |
| Se muestra el precio más alto de la acción seleccionado. | |

| **Requerimiento funcional 2.** | |
|---|---|
| **Nombre** | Consultar el precio más bajo de una acción |
| **Resumen** | El programa permitirá consultar el precio más bajo de un mercado de divisas en un rango de tiempo. |
| **Entrada** | |
| Nombre de la acción a consultar | |
| **Resultado** | |
| Se muestra el precio más bajo de la acción seleccionado. | |

| **Requerimiento funcional 3.** | |
|---|---|
| **Nombre** | Consultar el periodo de tiempo |
| **Resumen** | El programa permitirá consultar el periodo de tiempo en que un mercado de divisas tuvo su mayor crecimiento. |
| **Entrada** | |
| Nombre de la acción a consultar | |
| **Resultado** | |
| Se muestra el periodo de tiempo de la acción. | |

| **Requerimiento funcional 4.** | |
|---|---|
| **Nombre** | Mostrar una gráfica del estado de los precios de una acción |
| **Resumen** | El programa permitirá visualizar el estado de un mercado de divisas por medio de una gráfica. |
| **Entrada** | |
| Nombre de máximo 3 acciones | |
| **Resultado** | |
| Gráfica con el estado de los mercados de divisas | |

| **Requerimiento funcional 5.** | |
|---|---|
| **Nombre** | Mostrar qué acciones superan un valor |
| **Resumen** | El programa mostrará qué acciones superan un valor en un rango de tiempo determinado. |
| **Entrada** | |
| Valor a consultar y periodo de tiempo deseado. | |
| **Resultado** | |
| Nombre de las acciones que superan el valor consultado. | |

| **Requerimiento funcional 6.** | |
|---|---|
| **Nombre** | Mostrar cuáles acciones presentan mayor crecimiento |
| **Resumen** | El programa mostrará qué acciones presentaron mayor crecimiento en un rango de tiempo determinado. |
| **Entrada** | |
| Rango de tiempo | |
| **Resultado** | |
| Nombre de las acciones que tuvieron mayor crecimiento | |

## ***Diseño de casos de casos de pruebas unitarias:***

| **Prueba No. 1** | **Objetivo de la prueba:** Comprobar que el método addNode agrega ordenadamente los valores al árbol AVL y lo equilibre.<br>**Firma del método:** public void addNode(K key, V value) | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| DennAVLTree | addNode(K key, V value) | avl = new DennAVLTree<>();<br>nList = new int[7];<br>nList[0] = 8;<br>nList[1] = 3;<br>nList[2] = 12;<br>nList[3] =9;<br>nList[4]= 15;<br>nList[5]=2;<br>nList[6]= 6; | Add(nList[i], nList[i])<br>nList[i], es la llave que identifica al objeto.<br>nList[i], es el objeto a guardar.<br>Donde 0≤i<7 | Root = 8<br>Root.left = 3<br>Root.left.left = 2<br>Root.left.rigth = 6<br>Root.Rigth = 12<br>Root.rigth.left = 9<br>Root.rigth.rigth = 15<br>y está balanceado. |

| **Prueba No. 2** | **Objetivo de la prueba:** Comprobar que el método SearchNode busca los valores en el árbol AVL retorna null si la llave no está.<br>**Firma del método:** public DennNode<K, V> searchNode(K key) | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| DennBST | searchNode (K key, V value) | avl = new DennAVLTree<>();<br>nList = new int[7];<br>nList[0] = 8;<br>nList[1] = 3;<br>nList[2] = 12;<br>nList[3] =9;<br>nList[4]= 15;<br>nList[5]=2;<br>nList[6]= 6; | searchNode(nList[i])<br>nList[i], es la llave a buscar.<br>Donde 0≤i<7 | Se busca la llave nList[i] en el árbol AVL y se verifica que sea el mismo objeto.<br>Si no encuentra la llave, retornará null.<br>Donde 0≤i<7 |

| **Prueba No. 3** | **Objetivo de la prueba:** Comprobar que el método deleteNode elimina el objeto que corresponde a la llave del árbol AVL<br>**Firma del método:** public DennNode<K, V> deleteNode(K key) | | | |
|---|---|---|---|---|
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| DennAVL | deleteNode(K key) | avl = new DennAVLTree< >();<br>nList = new int[7];<br>nList[0] = 8;<br>nList[1] = 3;<br>nList[2] = 12;<br>nList[3] =9;<br>nList[4]= 15;<br>nList[5]=2;<br>nList[6]= 6; | deleteNode(nList[i])<br>Donde 0≤i<7 | **deleteNode**(8) entonces al buscarlo retornará null. Como 9 será la nueva raíz y al eliminarla **deleteNode**(9) entonces al buscarlo retornará null. La nueva raíz será la llave 12 con hijo derecho 15 e hijo izquierdo 3. |

| **Prueba No. 4** | **Objetivo de la prueba:** Comprobar que el método min busca el elemento menor en el árbol.<br>**Firma del método:** public DennNode<K, V> min(DennNode<K, V> d) | | | |
|---|---|---|---|---|
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| DennBST | min(DennNode <K, V> d) | avl = new DennAVLTree< >();<br>nList = new int[7];<br>nList[0] = 8;<br>nList[1] = 3;<br>nList[2] = 12;<br>nList[3] =9;<br>nList[4]= 15;<br>nList[5]=2;<br>nList[6]= 6; | min(DennNode <K, V> d)<br>Donde d es un nodo del árbol avl | El valor mínimo del hijo derecho de la raíz, min(12), es igual a 9 Ahora si eliminar el valor 9, el valor mínimo será él mismo, es decir el valor mínimo del hijo derecho 12 es 12. |

| **Prueba No. 5** | **Objetivo de la prueba:** Comprobar que el método successor busca un nodo sucesor en el árbol AVL<br>**Firma del método:** public DennNode<K, V> successor(DennNode<K,V> x) | | | |
|---|---|---|---|---|
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| DennBST | successor(DennNode<K,V> x) | avl = new DennAVLTree<>();<br>nList = new int[7];<br>nList[0] = 8;<br>nList[1] = 3;<br>nList[2] = 12;<br>nList[3] =9;<br>nList[4]= 15;<br>nList[5]=2;<br>nList[6]= 6; | successor(DennNode<K,V> x)<br>Donde x es el nodo al que se le busca su sucesor. | El valor sucesor del hijo derecho de la raíz, successor(12), es igual a 9<br>Ahora si eliminar el valor 9, el valor sucesor será él mismo, es decir el valor sucesor del hijo derecho 12 es 12. |

| **Prueba No. 6** | **Objetivo de la prueba:** Comprobar que el método rotationToLeft rota un nodo del árbol hacia la izquierda<br>**Firma del método:** public void rotationToLeft(DennNode<K, V> x) | | | |
|---|---|---|---|---|
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| DennBST | rotationToLeft(DennNode<K, V> x) | avl = new DennAVLTree<>();<br>nList = new int[7];<br>nList[0] = 8;<br>nList[1] = 3;<br>nList[2] = 12;<br>nList[3] =9;<br>nList[4]= 15;<br>nList[5]=2;<br>nList[6]= 6; | rotationToLeft(DennNode<K, V> x)<br>Donde x es el nodo a rotar. | El nodo con llave 3 tiene dos hijos, hijo izquierdo con llave 2 e hijo derecho con llave 6. Al rotar el nodo 3, quedaría el nodo 6 con hijo izquierdo 3 e hijo derecho null |

| **Prueba No. 7** | **Objetivo de la prueba:** Comprobar que el método rotationToRight rota un nodo del árbol hacia la derecha<br>**Firma del método:** public void rigthRotate(DennNode<K, V> x) | | | |
|---|---|---|---|---|
| **Clase para probar** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
| DennBST | rotationToRight (DennNode<K, V> x) | avl = new DennAVLTree< >();<br>nList = new int[7];<br>nList[0] = 8;<br>nList[1] = 3;<br>nList[2] = 12;<br>nList[3] =9;<br>nList[4]= 15;<br>nList[5]=2;<br>nList[6]= 6; | rotationToRight( DennNode<K, V> x)<br>Donde x es el nodo a rotar. | El nodo con llave 12 tiene dos hijos, hijo izquierdo con llave 9 e hijo derecho con llave 15. Al rotar el nodo 12, quedaría el nodo 9 con hijo derecho 12 e hijo izquierdo null. |

## 7. ***Implementación:***

Link del repositorio en github: https://github.com/dennvm09/StockExchange_Lab3