**METODO DE LA INGENIERÍA**

1. **Contexto problemático**

El mercado de las acciones y las divisas ha crecido exponencialmente a lo largo del tiempo, así mismo, ha ido evolucionando la tecnología a través de la historia, es por eso que ha surgido la necesidad de innovar respecto a la forma como se visualiza y se maneja el mercado de las acciones y las divisas por medio de la tecnología. Sin embargo la bolsa de valores de Colombia (BVC) actualmente no permite transar con acciones internacionales, sin embargo, aprovechando esta coyuntura el BVC quiere consolidar una aplicación. En el cual se desea desarrollar un programa que permita realizar análisis sobre los datos de acciones y divisas, donde se pueda ingresar datos de gran tamaño y se puedan reconocer patrones a cerca de estos mercados. Esta aplicación debe permitir el ingreso de datos, ya sea de manera masiva (archivos de texto plano, por ejemplo) o a través de una interfaz; eliminar o modificar datos; realizar consultas de mercados de capitales y mercados de divisas. Y donde finalmente se puedan consultar el precio más alto o más bajo de una divisa o acción en un intervalo de tiempo, el periodo de tiempo donde una acción o divisa tuvo su mayor o menor crecimiento y una serie de graficas que reflejen el precio de la acción o la divisa. Estos requerimientos puntuales, están mejor detallados en las tablas de Requerimientos funcionales, dentro de este documento.

**REQUERIMIENTOS FUNCIONALES**

|  |  |
| --- | --- |
| **NOMBRE** | Leer grandes cantidades de datos **P001** |
| **RESUMEN** | El sistema debe estar en la capacidad de leer gran cantidad de datos ya sea por medio de archivos de texto plano, pero que sea de manera masiva |
| **ENTRADA** | Una gran cantidad de información acerca de una mercado de acciones o un mercado de divisas |
| **SALIDA** |  |

|  |  |
| --- | --- |
| **NOMBRE** | Eliminar un mercado de acciones o de divisas **P002** |
| **RESUMEN** | El sistema debe estar en la capacidad de eliminar en su totalidad un mercado de acciones o de divisas |
| **ENTRADA** | El accionar de un Actionevent o botón donde se confirme que se va a borrar ese mercado |
| **SALIDA** | El programa sin el marcado anteriormente eliminado |

|  |  |
| --- | --- |
| **NOMBRE** | Modificar un mercado de acciones o de divisas **P003** |
| **RESUMEN** | El sistema debe estar en la capacidad de modificar el estado de una acción o una divisa, teniendo en cuenta cada uno de sus atributos |
| **ENTRADA** | * Fecha * Precio medio * Nombre |
| **SALIDA** | El programa está en la capacidad de modificar el estado del mercado |

|  |  |
| --- | --- |
| **NOMBRE** | Consultar el precio más alto de una acción en un rango de tiempo **P004** |
| **RESUMEN** | El sistema de estar en capacidad de permitir que el usuario consulte el precio más alto de una acción o de un mercado de divisas dado un rengo de tiempo. Eso quiere decir que el sistema permite consultar el precio más alto de la misma dada una fecha inicial y final. |
| **ENTRADA** | * Un valor que indique la fecha inicial * Un valor que indique la fecha final |
| **SALIDA** | El sistema muestra el precio más alto de una acción o de un mercado de divisas. |

|  |  |
| --- | --- |
| **NOMBRE** | Consultar el precio más bajo de una acción en un rango de tiempo **P005** |
| **RESUMEN** | El sistema de estar en capacidad de permitir que el usuario consulte el precio más bajo de una acción o de un mercado de divisas dado un rengo de tiempo. Eso quiere decir que el sistema permite consultar el precio más alto de la misma dada una fecha inicial y final. |
| **ENTRADA** | * Un valor que indique la fecha inicial   Un valor que indique la fecha final |
| **SALIDA** | El sistema muestra el precio más bajo de una acción o de un mercado de divisas. |

|  |  |
| --- | --- |
| **NOMBRE** | Consultar el periodo de tiempo donde una acción tuvo su mayor crecimiento **P006** |
| **RESUMEN** | El sistema debe estar en capacidad de permitir que el usuario consulta el periodo de tiempo donde una acción o mercado de divisas tuvo su mayor crecimiento. |
| **ENTRADA** |  |
| **SALIDA** | El sistema muestra el periodo de tiempo donde una acción o mercado de divisas tiene su mayor crecimiento. |

|  |  |
| --- | --- |
| **NOMBRE** | Mostrar en una gráfica el estado de una acción **P007** |
| **RESUMEN** | El sistema debe estar en capacidad de mostrar en una gráfica el estado de una acción o de un mercado de divisas, permitiendo tener allí máximo 3 acciones o mercados de divisas en donde cada indicador de gráfica deberá tener un color diferente. |
| **ENTRADA** |  |
| **SALIDA** | El sistema muestra una gráfica con los estados de unas acciones o mercados de divisas. |

|  |  |
| --- | --- |
| **NOMBRE** | Mostrar las acciones o Mercado de divisas superan un valor en un rango de tiempo **P008** |
| **RESUMEN** | El sistema debe estar en capacidad de mostrar cuales acciones o mercado de divisas superan un valor en un rango determinado de tiempo, es decir un valor, dada una fecha inicial y una fecha final. |
| **ENTRADA** | * Valor que indica el valor superado * Valor que indica la fecha final * Valor que indica la fecha inicial |
| **SALIDA** | El sistema muestra cuales acciones o mercado de divisas superan un valor en un rango de tiempo. |

|  |  |
| --- | --- |
| **NOMBRE** | Muestra cuales 3 acciones o Mercados que presentaron mayor crecimiento en un rango de tiempo **P009** |
| **RESUMEN** | El sistema debe estar en capacidad de mostrar las 3 acciones o mercados que presentaron mayor crecimiento en un rango de tiempo, es decir que tuvieron mayor crecimiento en una fecha inicial dad y una fecha final dada. |
| **ENTRADA** | Valor que indica la fecha inicial  Valor que indica la fecha final |
| **SALIDA** | El sistema muestras las 3 acciones o mercados que presentaron mayor crecimiento dado un rango de tiempo. |

1. **Recopilación de la información**

Para la solución de este problema es fundamental tener conocimiento acerca de qué es un mercado de divisas y el mercado de acciones.

**Mercado de divisas**: El mercado de divisas o mercado de tipos de cambio es un mercado global y descentralizado en el que se negocian divisas y que nació con el objetivo de facilitar cobertura al flujo monetario que se deriva del comercio internacional. También conocido como Foreign Exchange o FX, el de divisas es un mercado fundamentalmente no organizado, lo que se denomina en inglés ‘Over The Counter‘ (OTC). Hoy en día es el mercado financiero más grande del mundo con un volumen medio diario negociado en torno a los 5 billones de dólares y con una importante y creciente “electronificación”.

**Funciones del mercado de divisas:**

* Fijar los precios de unas monedas con respecto a otras (pares de divisas), es decir, definir el número de unidades a entregar de una moneda a cambio de una unidad de otra moneda distinta.
* Permitir la realización de coberturas de riesgo de tipo de cambio, por ejemplo, cuando se realizan inversiones en moneda extranjera.
* Favorecer el intercambio de fondos entre diferentes países; nos encontramos con países con excedentes de liquidez y países que necesitan dicha liquidez.
* Financiar el comercio internacional, cuyas operaciones representan una parte muy significativa del mercado de divisas.

(Fernandez, 2017)

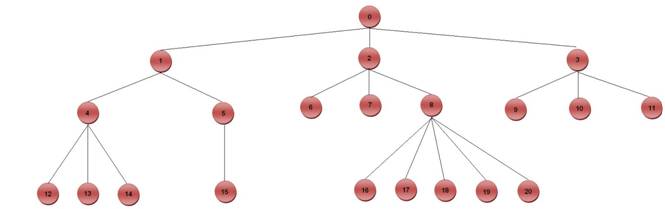
**Mercado de acciones:** Es aquel en el que los valores que se transan no tienen preestablecido el rendimiento, siendo las acciones el título característico de este mercado. La rentabilidad está ligada a las utilidades obtenidas por la empresa en la cual se invierte, así como a las posibles variaciones en los precios de los valores, dadas por las condiciones existentes en el mercado. (BVC Bolsa de valores de colombia, 2018)

**X-Stream de Nasdaq-OMX**: La Bolsa de Valores de Colombia administra el sistema transaccional para la negociación de Acciones, soportada en la plataforma de negociación X-Stream de Nasdaq-OMX. Maneja el comercio de miles de millones de valores por día en todo el mundo. Es el sistema de gestión de mercado más utilizado, operado por más de 25 intercambios, más que cualquier otra plataforma de negociación.

Además de estos conocimientos es importante tener en cuenta las estructuras y sus características que nos pueden ser útiles para la solución de este problema.

**Árbol n-ario:** Un árbol n-ario es una estructura recursiva, en la cual cada elemento tiene un número cualquiera de árboles n-arios asociados. Estos árboles corresponden a la generalización de un árbol binario. La diferencia radica en que esta estructura puede manejar múltiples subárboles asociados a cada elemento, y no solamente 2, como en el caso de los árboles binarios.

Los conceptos de padre, hijo, hermano, nivel, altura, peso, camino, etc que son utilizados en árboles binarios, pueden ser extendidos fácilmente a los árboles n-arios.

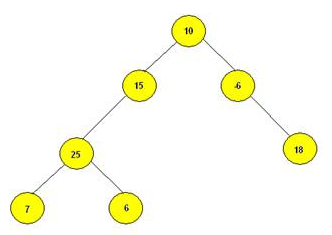


**Árbol binario:** Un árbol binario es una estructura de datos recursiva compuesta por un elemento (denominado raíz) y dos subárboles asociados, también binarios, denominados hijo izquierdo e hijo derecho.

Algunos conceptos importantes en árboles binarios se explican a continuación:

* Árbol vacío: árbol que no tiene elementos ni subárboles asociados.
* Hijo: un elemento B se denomina hijo de un elemento A si B es la raíz de alguno de los subárboles asociados con A. En este mismo caso se dice que A es Padrede B.
* Hermano: dos elementos B y C son hermanos si son hijos del mismo padre.
* Hoja: un elemento de un árbol binario es una hoja si sus hijos izquierdo y derecho son vacíos.
* Elemento no terminal: todo elemento que no sea una hoja se considera elemento no Terminal.
* Camino: un camino entre dos elementos A y B se define como una secuencia de la forma en la que el primer elemento es A y el último es B y cada elemento es padre de su sucesor.
* Longitud camino: la longitud de un camino es n-1.
* Rama: es un camino que inicia en la raíz y finaliza en una hoja.
* Peso: es el número de elementos que hay en el árbol.
* Altura: es la longitud del camino más largo desde la raíz más 1.
* Nivel de un elemento: es la longitud del camino que parte de la raíz y llega al elemento.
* Árbol binario completo: árbol en el que todo elemento no terminal tiene asociados exactamente dos subárboles no vacíos.
* Árbol binario lleno: árbol binario completo y que tiene además todas sus hojas al mismo nivel.
* Recorrido en preorden: consiste en visitar primero la raíz, luego el subárbol izquierdo en preorden y finalmente el subárbol derecho en preorden.
* Recorrido en postorden: consiste en visitar primero en postorden el subárbol izquierdo, luego el derecho y finalmente la raíz.
* Recorrido en inorden: consiste en visitar primero el subárbol izquierdo en inorden, luego la raíz y finalmente el subárbol derecho en inorden.
* Recorrido por niveles: consisten en visitar primero la raíz, luego los elementos del nivel 1 de izquierda a derecha y así sucesivamente hasta visitar todos los elementos del árbol.

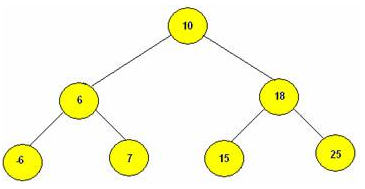
La complejidad de la búsqueda en los árboles binarios es O(n), donde n es el peso del árbol.  
Ejemplo:



**Árbol binario de búsqueda:** Un árbol binario ordenado es un árbol binario en el que todos los elementos del subárbol izquierdo son menores que la raíz y todos los elementos del subárbol derecho son mayores que la raíz. A su vez, los subárboles izquierdo y derecho son árboles binarios ordenados.

Una propiedad importante de los árboles binarios ordenados es que su recorrido en inorden resulta en la visita de sus elementos en orden ascendente.

Este tipo de estructuras sirven para el almacenamiento de elementos entre los que exista una relación de orden. La complejidad de la búsqueda en los árboles binarios ordenados en el peor de los casos es O(n), donde n es el peso del árbol. En el caso promedio, la complejidad de la búsqueda es O(log2 n).

Ejemplo:

(Universidad de los Andes, 2016)

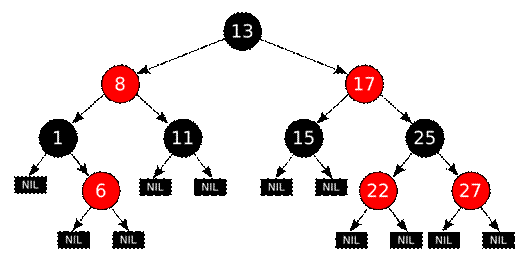
**Árbol rojo y negro:** Un árbol rojo negro es un árbol binario donde cada nodo tiene también un atributo de color, cuyo valor puede ser o rojo o negro. Las hojas de un árbol rojo negro son irrelevantes y no tienen datos.

Para que un árbol binario ordenado sea considerado como un árbol rojo negro debe cumplir con las siguientes propiedades:

* La raíz del árbol es negra.
* Los hijos de un nodo rojo son negros.
* Las hojas del árbol son negras.
* Todas las ramas del árbol (caminos desde la raíz hasta una hoja) tienen el mismo número de nodos negros.

Este tipo de estructuras sirven para el almacenamiento de elementos entre los que exista una relación de orden. La complejidad de la búsqueda en los árboles rojo negro es O(log2 n).

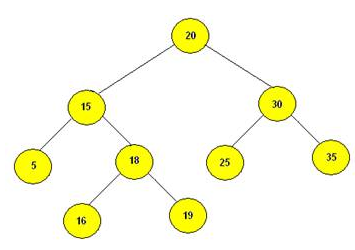
Ejemplo:



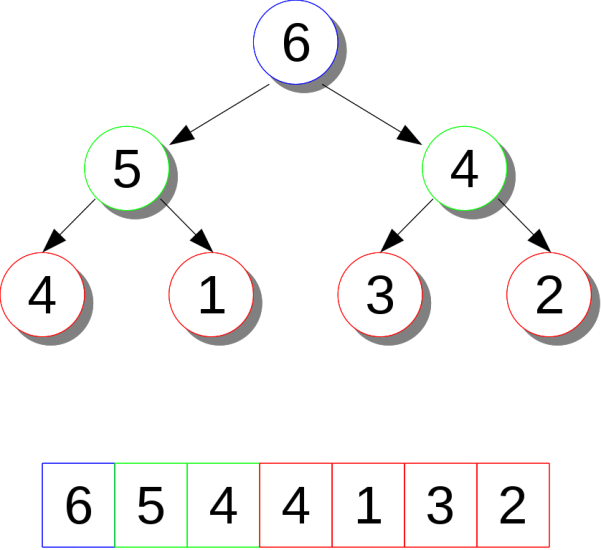
(Universidad de los Andes, 2016)

**Árbol AVL:** Un árbol AVL es un árbol binario ordenado y balanceado por altura en el que sus subárboles izquierdo y derecho son también AVL. La característica de balanceo implica que la altura de los dos subárboles asociados no puede diferir en más de uno. Gracias a esto se garantiza que la complejidad de la búsqueda en los árboles AVL en el peor de los casos sea O(log2 n).

Los árboles AVL toman su nombre de los apellidos de sus inventores Adelson- Velskii y Landis, quienes dieron a conocer el concepto de los mismos en su publicación de 1962 titulada "An algorithm for the organization of information".

Ejemplo:

(Universidad de los Andes, 2016)

**Montículo**: Un montículo es una estructura de datos que cumple con las condiciones de un árbol binario balanceado, pero que se implementa en un array que permite conservar estas propiedades. La siguiente es una representación gráfica de un heap:

(Universidad de los Andes, 2016)

**Tabla hash:** Una tabla de Hash es una estructura de acceso directo, en la que cada elemento tiene asociado una llave. La llave de un elemento es un identificador que tiene significado en el contexto del problema en el que esté siendo utilizada la tabla. Al interior de la tabla las llaves son únicas.

En una tabla de Hash no existe noción de orden tanto para los elementos como para sus respectivas llaves. Los elementos no tienen predecesores ni sucesores. No es posible realizar recorrido alguno sobre las llaves ni elementos. La única forma de acceder a los elementos de la tabla es mediante sus llaves asociadas. La ventaja principal de la tabla de Hash se encuentra en su efectividad para consultar los elementos a través de su llave de forma directa.

Otros conceptos importantes relacionados con la tabla de Hash se presentan a continuación:

**Tamaño**: es el número de elementos presentes en la tabla.

**Espacio de llaves:** es el conjunto de todas las llaves que pueden existir.

**Área primaria:** es el espacio físico en el que se colocan los elementos de la estructura.

**Dirección de un elemento**: es la posición que un elemento ocupa en el área primaria.

**Capacidad:** es el tamaño del área primaria.

**Factor de carga:** es el tamaño de la tabla sobre su capacidad. Indica que tan saturada la tabla se encuentra.

(Universidad de los Andes, 2016)

**Tipos de Datos Abstractos**: Se diferencian de los tipos de datos en el sentido de que los datos abstractos son especificados de manera precisas y diseñados independientemente de cualquier implementación, es decir que pueden ser implementados en cualquier lenguaje de programación ya que su forma de definirlos proporciona la información necesaria para hacerlo. El siguiente es un ejemplo de cómo se representa un tipo de dato abstracto de manera formal:

****

1. **Búsqueda de soluciones creativas**

Existen numerosas estructuras de datos que se encargan de almacenar información de una manera particular, haciendo que cada una de ellas tenga diferentes características que pueden adaptarse de mejor manera a cierto tipo de problemas dependiendo de las condiciones y necesidades que proporcione el mismo.

Teniendo en cuenta lo anterior, se procederá a hacer uso de una técnica conocida como lluvia de ideas para enlistar algunas de las opciones que a primera vista son factibles para resolver de manera pertinente el problema abordado:

**Alternativa 1: Tabla Hash**

Esta es una estructura de datos que se caracteriza porque funciones como buscar o insertar son notablemente rápidos, haciendo que sea una excelente alternativa para solucionar algunos aspectos del problema ya que se requiere un nivel de eficiencia aceptable en los algoritmos y estructuras usadas en la solución del problema.

**Alternativa 2: Montículos**

**“**La estructura de datos conocida como montículo es un arreglo de objetos que podemos ver como un árbol binario casi completo. Cada nodo del árbol corresponde a un elemento del arreglo” (Thomas H. Cormen, 2009)**.** Esta estructura de datos también podría incluirse en el grupo de candidatos a suplir la funcionalidad de almacenar los precios de las divisas, aunque dada la gran cantidad de datos a almacenar, puede resultar ineficiente el uso de esta estructura.

**Alternativa 3: Árbol Binario**

“Un árbol binario es un conjunto finito de elementos, el cual está vacío o dividido en tres subconjuntos separados: El primer subconjunto contiene un elemento único llamado raíz del árbol. El segundo subconjunto es en sí mismo un árbol binario y se le conoce como subárbol izquierdo del árbol original. El tercer subconjunto es también un árbol binario y se le conoce como subárbol derecho del árbol original“ (Serrano, 2013). Este tipo de estructura es una fuerte candidata debido a que sus operaciones de búsqueda, eliminación e inserción son notablemente rápidas. La desventaja de esta estructura de datos es que no es auto-balanceada y ello puede traer problemas en la eficiencia de las operaciones elementales.

**Alternativa 4: Árbol n-ario**

“Un árbol n-ario es una estructura recursiva, en la cual cada elemento tiene un número cualquiera de árboles n-arios asociados. Estos árboles corresponden a la generalización de un árbol binario. La diferencia radica en que esta estructura puede manejar múltiples subárboles asociados a cada elemento, y no solamente 2, como en el caso de los árboles binarios.” (Blanco, 2017). Esta estructura de datos es poco viable para el problema que se desea resolver, debido a que esta estructura no permite mantener un criterio de orden determinado y esa es la principal condición del software a crear.

**Alternativa 5: Arboles Rojinegros.**

“Un árbol rojo-negro es un árbol de búsqueda binario con un bit extra de almacenamiento por nodo: Su color, que puede ser rojo o negro” (Thomas H. Cormen, 2009). Esta es una estructura de datos idéntica a un árbol binario pero con ciertas características adicionales que le permiten tener la facultad de auto-balancearse por sí solos. Esta es una alternativa muy viable debido a que puede soportar las dos principales condiciones del problema en cuestión: Un criterio especifico de orden y una notable eficiencia en las operaciones de los elementos de la estructura.

**Alternativa 6: Arboles AVL.**

“Un árbol AVL es un árbol binario de búsqueda con altura equilibrada: Para cada nodo x, las alturas de los subárboles izquierdos y derechos de x difieren como máximo 1.” (Thomas H. Cormen, 2009). Esta también resulta una alternativa a tener en cuenta de cara a la resolución del problema en cuestión debido a su condición de auto-balanceo, permitiendo que la estructura se mantenga equilibrada y que sea posible mantener una notoria eficiencia en las operaciones principales que manipulan los elementos almacenados.

1. **Transición de ideas y diseños preliminares**

A continuación se evaluarán individualmente cada una de las alternativas planteadas en el inciso anterior y se procederá a descartar las propuestas menos factibles teniendo en cuenta los requerimientos del problema abordado, es decir, sus funcionalidades y condiciones de eficiencia. Para ello se procederá a aplicar la técnica de revisión selectiva que consiste en describir los aspectos relevantes de la información abordada dividiéndolos en dos secciones conocidas como ventajas y desventajas:

|  |  |  |
| --- | --- | --- |
| **Alternativa** | **Ventajas** | **Desventajas** |
| 1. **Tabla Hash** | Los algoritmos que manipulan este tipo de estructura son muy eficientes dado que tienen una complejidad temporal O(1). | El proceso de redimensionamiento de la tabla resulta una maniobra costosa de implementar. Además, no es posible establecer un criterio de ordenamiento de los elementos almacenados. |
| 1. **Montículo** | Los objetos guardados pueden ser organizados de manera ascendente o descendente. La ordenación se realiza con una complejidad temporal relativamente óptima. | La complejidad temporal puede llegar a ser ineficiente si se trata de un número grande de elementos o nodos, hecho que hace imposible la implementación de esta estructura en el problema en cuestión. |
| 1. **Árbol Binario** | Su funcionamiento puede emparejarse directamente con situaciones de la vida real. La información puede guardarse de manera ordenada. Puede redimensionarse sin mayor problema. | Si un árbol no está equilibrado, sus funcionalidades pueden no ser eficientes. No es posible usarlo para datos que no requieran un orden. Además no cumple con la propiedad de auto-balanceo. |
| 1. **Árbol n-ario** | Su funcionamiento puede emparejarse directamente con situaciones de la vida real. Puede redimensionarse sin mayor problema. | No es posible guardar información de manera ordenada, aunque si de manera jerárquica. |
| 1. **Árbol Rojinegro** | Su principal ventaja es que puede auto-balancearse, además de que es posible establecer un criterio de orden entre los elementos. | Las operaciones de eliminación e inserción suelen ser difíciles de entender y por ende de implementar en código. |
| 1. **Árbol AVL** | Su principal ventaja es que puede auto-balancearse, además de que es posible establecer un criterio de orden entre los elementos. | Las operaciones de eliminación e inserción suelen ser difíciles de entender y por ende de implementar. |

Teniendo en cuenta el análisis anterior, resulta necesario descartar las estructuras de datos cuyo funcionamiento no proporciona una solución óptima al problema planteado. Este es el caso del montículo debido a que la complejidad temporal de sus algoritmos es elevada comparada con otras estructuras, además de que este problema puede tornarse más grave al tener la necesidad de manipularse una gran cantidad de datos.

A continuación se expondrán los criterios mediante los cuales se elegirán las estructuras de datos a utilizar en la resolución del problema en cuestión:

**Criterio 1:** Complejidad temporal de los algoritmos que manipulan la estructura de datos.

|  |  |
| --- | --- |
| **Complejidad Temporal** | **Puntuación** |
| O (logn) | 4 |
| O (n+k) | 2 |
| O (1) | 6 |

**Criterio 2:** Criterio de ordenamiento entre sus elementos.

|  |  |
| --- | --- |
| **Orden entre sus elementos** | **Puntuación** |
| Si | 10 |
| No | 5 |

**Criterio 3:** Propiedad de auto-balanceo de la estructura de datos.

|  |  |
| --- | --- |
| **Propiedad de auto-balanceado** | **Puntuación** |
| Auto-balanceado | 10 |
| No auto-balanceado | 2 |

En el siguiente recuadro se mostraran las diferentes alternativas que, luego de evaluarlas con los criterios descritos, se elegirán las que cumplan con las condiciones establecidas con mayor rigurosidad.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Criterio 1** | **Criterio 2** | **Criterio 3** | **Total** |
| Alternativa 1. Tabla Hash | 6 | 5 | 2 | 13 |
| Alternativa 2. Árbol Binario | 4 | 10 | 2 | 16 |
| Alternativa 3: Árbol n-ario | 4 | 5 | 2 | 11 |
| Alternativa 4. Árbol rojinegro | 4 | 10 | 10 | 24 |
| Alternativa 5. Árbol AVL | 4 | 10 | 10 | 24 |

**Selección definitiva:** Teniendo en cuenta el método de selección utilizado anteriormente, se llegó a la conclusión de que las estructuras que deberán ser usadas en la creación del software son las siguientes: Árbol ARN y árbol AVL. La principal razón por la que se eligieron estas estructuras es porque su complejidad temporal es baja y además porque es posible mantener un criterio de ordenamiento entre sus elementos.

**6. Preparación de informes y especificaciones:**

**Especificación del problema:**

**Problema:** Desarrollar un programa que permita administrar la información del mercado de divisas y de acciones de unos determinados archivos (datasets) suministrados al programa.

**Entrada:** Necesidades a visualizar por el usuario, intervalos de tiempo, precios, entre otros  **Salidas:** Información detallada de cada jugador de la liga, incluyendo sus datos y estadísticas deportivas.

Como parte de las especificaciones de las estructuras de datos utilizadas para resolver el problema se definieron los TADs de cada una de ellas:

|  |
| --- |
| **TAD Árbol Rojo-Negro** |
|  |
| X es el conjunto domino de los valores con un tipo de dato particular  U es el universo de llaves |
| * CrearARN() -> <Arbol Rojo-Negro> * Insertar (U, X) * Buscar (U) -> <X> * Eliminar (U) |

|  |
| --- |
| **TAD Árbol AVL** |
|  |
| X es el conjunto domino de los valores con un tipo de dato particular  U es el universo de llaves |
| * CrearAVL() -> <AVL> * Insertar (U, X) * Buscar (U) -> <X> * Eliminar (U) |

|  |
| --- |
| **Insertar (U, X)** |
| \*Agrega un nuevo elemento al árbol, lo ubica de acuerdo a un criterio de orden específico y se balancea la estructura si es necesario.  {pre: La clave debe existir dentro del universo de llaves}  {post: Se agregó un elemento a al árbol de acuerdo al criterio especificado} |

|  |
| --- |
| **Buscar (U)** |
| \*Busca el elemento dentro del árbol de acuerdo a un criterio de búsqueda especifico o usando la clave asociada al elemento en cuestión.  {pre: root!=null}  {post: Se retornó un elemento con la clave o el criterio de búsqueda ingresado como parámetro. } |

|  |
| --- |
| **Eliminar (U)** |
| \*Elimina el elemento del árbol que corresponda a la clave o el criterio U de búsqueda.  {pre: existe }  {post: Se eliminó el objeto con el criterio o clave ingresada como } |

|  |
| --- |
| **Obtener (criterio)** |
| \*Retorna el elemento al que le pertenece el criterio de búsqueda ingresado como parámetro.  {pre: }  {post: Se ha encontrado el elemento con el criterio ingresado como parámetro} |

|  |
| --- |
| **Balancear (subárbol)** |
| \*Retorna el subárbol ingresado ya balanceado.  {pre: null}  {post: Se balanceó el subárbol ingresado como parámetro} |

|  |
| --- |
| **ObtenerColor ()** |
| \*Retorna el color de un nodo determinado.  {pre: null}  {post: Se balanceó el subárbol ingresado como parámetro} |

|  |
| --- |
| **estaBalanceado ()** |
| \*Retorna verdadero om.  {pre: null}  {post: Se balanceó el subárbol ingresado como parámetro} |

|  |
| --- |
| **estaBalanceado ()** |
| \*Retorna verdadero om.  {pre: null}  {post: Se balanceó el subárbol ingresado como parámetro} |

|  |
| --- |
| **cambiarColor ()** |
| \* Se modificaron los colores de los nodos de un árbol Rojo-Negro de acuerdo a las condiciones propias de este tipo de estructura.  {pre: null}  {post: Se modificaron los colores de los nodos de un árbol Rojo-Negro} |

|  |
| --- |
| **rotarIzq (Nodo)** |
| \*Se rota hacia la izquierda el subárbol cuya raíz es el nodo ingresado como parámetro.  {pre: null}  {post: Se roto el subárbol hacia la izquierda} |

|  |
| --- |
| **rotarDer (Nodo)** |
| \*Se rota hacia la izquierda el subárbol cuya raíz es el nodo ingresado como parámetro.  {pre: null}  {post: Se roto el subárbol hacia la derecha} |

1. **Implementación del diseño**

A continuación se presentara una lista de las tareas a implementar por el programa:

1. Ingresar información del mercado de acciones y de divisas.
2. Eliminar información del mercado de acciones o de divisas.
3. Consultar información de un mercado de acciones o de divisas.
4. Mostrar información de un mercado de acciones o de divisas.

|  |  |
| --- | --- |
| Especificación de subrutinas | Implementación |
| |  |  | | --- | --- | | **Nombre** | add | | **Descripción** | Ese método se encarga de añadir un nuevo mercado de divisas | | **Entrada** | Información del nuevo jugador a añadir. | | **Retorno** | Sin salida. | | **public** **boolean** insert(K key, V value) {  size++;  **if** (root == **null**) {  root = **new** AVLNode<>(key, value, **null**);  **return** **true**;  }  AVLNode<K,V> n = root;  **while** (**true**) {  AVLNode<K,V> parent = n;  **boolean** goLeft = n.getKey().compareTo(key)>0;  n = goLeft ? n.getLeft() : n.getRight();  **if** (n == **null**) {  **if** (goLeft) {  parent.setLeft(**new** AVLNode<>(key, value, parent));  } **else** {  parent.setRight(**new** AVLNode<>(key, value, parent));  }  rebalance(parent);  **break**;  }  }  **return** **true**;  } |
| |  |  | | --- | --- | | **Nombre** | delete | | **Descripción** | Este método se encarga de eliminar un mercado de | | **Entrada** | Clave del jugador a eliminar. | | **Retorno** | Sin salida. | | @Override  **public** **void** delete(K delKey) {  **if** (root == **null**)  **return**;  size--;  AVLNode<K,V> child = root;  **while** (child != **null**) {  AVLNode<K,V> node = child;  child = delKey.compareTo(node.getKey()) >= 0 ? node.getRight() : node.getLeft();  **if** (delKey.equals(node.getKey())) {  delete(node);  **return**;  }  }  size++;  }  NOTA: la continuación de este método se puede encontrar |
| |  |  | | --- | --- | | **Nombre** | balance | | **Descripción** | Este método se encarga de balancear el subárbol ingresado como parámetro. | | **Entrada** | Subárbol. | | **Retorno** | Subárbol balanceado. | | **private** **void** rebalance(AVLNode<K,V> n) {  setBalance(n);  **if** (n.getBalance() == -2) {  **if** (height(n.getLeft().getLeft()) >= height(n.getLeft().getRight()))  n = rotateRight(n);  **else**  n = rotateLeftThenRight(n);  } **else** **if** (n.getBalance() == 2) {  **if** (height(n.getRight().getRight()) >= height(n.getRight().getLeft()))  n = rotateLeft(n);  **else**  n = rotateRightThenLeft(n);  }  **if** (n.getParent() != **null**) {  rebalance(n.getParent());  } **else** {  root = n;  }  } |
| |  |  | | --- | --- | | **Nombre** | rotateLeft | | **Descripción** | Este método se encarga de rotar hacia la izquierda el subárbol ingresado como parámetro. | | **Entrada** | Raíz del subárbol a rotar | | **Retorno** | Sin salida | | **private** AVLNode<K,V> rotateLeft(AVLNode<K,V> a) {  AVLNode<K,V> b = a.getRight();  b.setParent(a.getParent());  a.setRight(b.getLeft());  **if** (a.getRight() != **null**)  a.getRight().setParent(a);  b.setLeft(a);  a.setParent(b);  **if** (b.getParent() != **null**) {  **if** (b.getParent().getRight() == a) {  b.getParent().setRight(b);  } **else** {  b.getParent().setLeft(b);  }  }  setBalance(a, b);  **return** b;  } |

**NOTA:** Para el desarrollo de la estructura Árbol rojo y negro me apoyé en el árbol implementado por Zarar Siddiqi**,** el cual se puede observar en su enlace de git hub. <https://github.com/Arsenalist/Red-Black-Tree-Java-Implementation>

**BIBLIOGRAFÍA**

Asencio, A. (2011). *iuma.* Obtenido de http://www.iuma.ulpgc.es

Blanco, O. (2 de 12 de 2017). *oscarblancarteblog.* Obtenido de https://www.oscarblancarteblog.com/

BVC Bolsa de valores de colombia. (12 de 4 de 2018). *Mercado de Renta Variable*. Obtenido de https://www.bvc.com.co/pps/tibco/portalbvc/Home/Mercados/descripciongeneral/acciones?action=dummy

Castrillon, C. (2015). *ocw.* Obtenido de ocw.opm.es

Fernandez, F. J. (26 de 05 de 2017). *El mercado de divisas: ¿Qué es y cómo funciona?* Obtenido de https://www.bbva.com/es/mercado-divisas-que-es-como-funciona/

Heileman, G. L. (1994). *HeEstructuras de Datos, Algoritmos y Programación Orientada a Objetos.* Madrid: McGraw-Hill.

S., J. A. (2006). *Introduccion a las Estructuras de Datos.* Bogota, Colombia: Pearson.

Serrano, M. (2013). Obtenido de https://www.infor.uva.es/~mserrano/EDI/cap5.pdf

Thomas H. Cormen, C. E. (2009). *Introduction to Alogrithms.* Londres: Massachusetts Institute of Technology.

*uc3m*. (02 de 06 de 2017). Obtenido de http://www.it.uc3m.es/

Universidad de los Andes. (2016). Obtenido de https://cupintranet.virtual.uniandes.edu.co/sitio/index.php/cursos/estructuras-de-datos/cupi2collections/estructuras-de-datos

*Wikipedia*. (2013).

**ANALISIS PESTLE**

**DISEÑO DE PRUEBAS UNITARIAS AUTOMATICAS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CLASE** | **METODO** | **ESCENARIO** | **VALORES DE ENTRADA** | **SALIDA** |
| BST | insertTest | Se inicializa un árbol de tipo BST | Agrega El número máximo de nodos | Todos los nodos fueron agregados y no están duplicados |
| BST | deleteTest | Se inicializa un árbol de tipo BST | Agrega el número máximo de nodos | Se comprueba que todos los nodos hayan sido eliminados |
| BST | searchTest | Se inicializa un árbol de tipo BST | Se agrega un numero x de nodo y luego un número y de nodos lo cual van a indicar el intervalo de búsqueda | De vuelve el valor de la llave a buscar |
| BST | keysInOrderTest | Se inicializa un árbol de tipo BST | Se agrega el número máximo de nodos | Se verifica que una lista de llaves de nodos este ordenado en el orden natural |
| BST | valuesInOrder | Se inicializa un árbol de tipo BST | Se agrega el nuero máximo de nodos | Se verifica que los valores correspondan a las llaves en orden |
| AVL | insertTest | Se inicializa un árbol de tipo AVL | Agrega El número máximo de nodos | Todos los nodos fueron agregados y no están duplicados |
| AVL | deleteTest | Se inicializa un árbol de tipo AVL | Agrega el número máximo de nodos | Se comprueba que todos los nodos hayan sido eliminados |
| AVL | searchTest | Se inicializa un árbol de tipo AVL | Se agrega un numero x de nodo y luego un número y de nodos lo cual van a indicar el intervalo de búsqueda | De vuelve el valor de la llave a buscar |
| AVL | keysInOrderTest | Se inicializa un árbol de tipo AVL | Se agrega el número máximo de nodos | Se verifica que una lista de llaves de nodos este ordenado en el orden natural |
| AVL | valuesInOrder | Se inicializa un árbol de tipo AVL | Se agrega el nuero máximo de nodos | Se verifica que los valores correspondan a las llaves en orden |
| RBT | insertTest | Se inicializa un árbol de tipo RBT | Agrega El número máximo de nodos | Todos los nodos fueron agregados y no están duplicados |
| RBT | deleteTest | Se inicializa un árbol de tipo RBT | Agrega el número máximo de nodos | Se comprueba que todos los nodos hayan sido eliminados |
| RBT | searchTest | Se inicializa un árbol de tipo RBT | Se agrega un numero x de nodo y luego un número y de nodos lo cual van a indicar el intervalo de búsqueda | De vuelve el valor de la llave a buscar |
| RBT | keysInOrderTest | Se inicializa un árbol de tipo RBT | Se agrega el número máximo de nodos | Se verifica que una lista de llaves de nodos este ordenado en el orden natural |
| RBT | valuesInOrder | Se inicializa un árbol de tipo RBT | Se agrega el nuero máximo de nodos | Se verifica que los valores correspondan a las llaves en orden |