Universidad Católica del Norte
Facultad de Ingeniería y Ciencias Geológicas
Departamento de Ingeniería de Sistemas y Computación

Taller N°3

Autores:
- Yeison Olivares E.
- Rodrigo Dominguez M.

Asignatura: Estructura de datos
Profesora: Angelica Flores B.
Ayudantes:
- Juan Quispe T.
- Fabian Rojas A.

Fecha: 03/06/2018

Índice

Introducción

En el presente informe se detalla el desarrollo del tercer taller evaluado de la asignatura estructura de datos, el cual contiene las bases para poder comprender el funcionamiento del programa realizado.

Objetivos del Taller

1. Realizar una documentación en informe sobre un programa realizado desde un enunciado, explicando la funcionalidad del mismo, incluyendo manual de usuario.
2. Realizar un programa en Visual Studio debidamente documentado incluyendo todo lo relevante para que se entienda la finalidad de la solución.
3. Dar una solución contemporánea aplicando árboles binarios tipo ABB, AVL.
4. Implementar los conceptos aprendidos en clases.

Contexto del taller

Jimmy era un estudiante de informática y vive en una ciudad dominada por la delincuencia. Los asaltos, violaciones y otros crímenes son pan de cada día.

Pero Jimmy no se queda con los brazos cruzados y decide abandonar sus estudios para ingresar a la escuela de detectives y acabar uno a uno con los delincuentes. Con esta meta en su vida de mejorar su ciudad nos solicita a nosotros diseñar un programa que almacene la información de todos los delincuentes que deambulan por la ciudad. Como Jimmy abandonó su carrera, él no lo puede hacer por su cuenta. Utilizando un archivo txt donde se almacena la información de los delincuentes, esto se explicará en la sección de manuel de usario.

Resumen del programa

Al momento de abrir el programa se despliega un menú con las siguientes opciones:

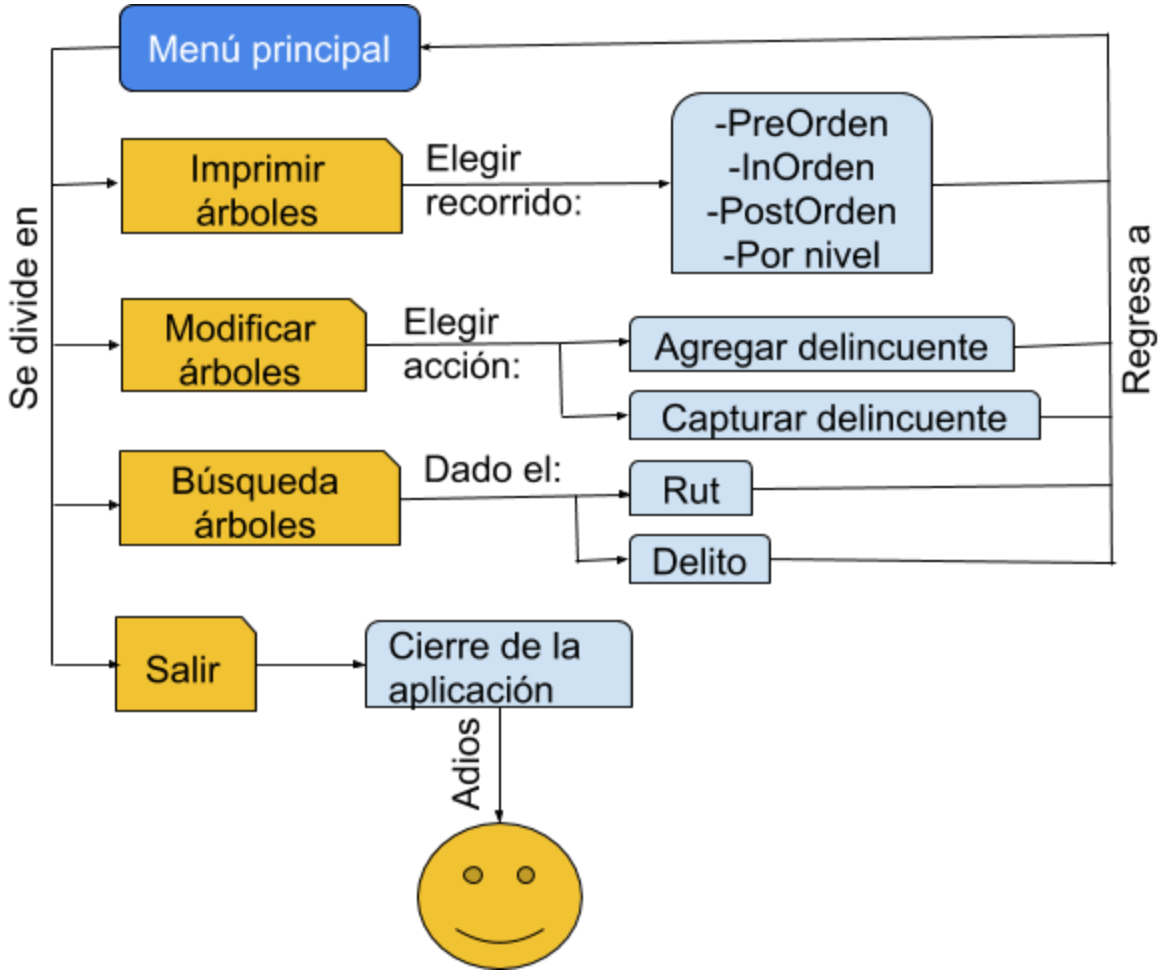
1. Imprimir árboles: Se imprimen ambos árboles siguiendo el tipo de recorrido especificado por el usuario.
2. Modificar árboles: Se puede agregar o eliminar la información de un delincuente.
3. Búsqueda en árboles: Se busca la información de un delincuente mediante su rut o un delito en específico.
4. Salir: Se cierra la aplicación.

Manual de usuario

El Informe será presentado de acuerdo a las fechas estipuladas en el enunciado al igual que el código fuente del programa como el manual de usuario.

El código será desarrollado en lenguaje C++ en la IDE Visuals Studio 2017.

Para crear la petición de Jimmy el programa consta de diversos menús para satisfacer cada requerimiento, los cuales son:



Uso del programa

Bienvenido en el presente párrafo se le explicará el funcionamiento e ingreso correcto de datos:

Antes de iniciar la aplicación verifique que exista el documento datos.txt con la información de cada delincuente y que tenga el siguiente formato para un correcto funcionamiento del programa:
Cada campo corresponde a: Rut, Nombre y apellido, Alias, Peligrosidad, delito
Ejemplo:

11222333,Dangelo Araya,care malo,alta,Hurto

1. Inicio de aplicación:

Se presentan el nombre de la aplicación, junto con 4 opciones a elegir.

a. Opción 1.
   i. Desplegar delincuentes: Se accede mediante el ingreso del número 1 por teclado y luego presionar la tecla enter, y se despliega un menú de opciones, véase punto 2 del manual.

b. Opción 2.
   i. Modificar árboles: Se accede mediante el ingreso del número 2 por teclado y luego presionar la tecla enter, y se despliega un menú de opciones, véase punto 3 del manual.

c. Opción 3.
   i. Búsqueda: Se accede mediante el ingreso del número 3 por teclado y luego presionar la tecla enter, y se despliega un menú de opciones, véase punto 4 del manual.

d. Opcion 4.
   i. Salir: Se accede mediante el ingreso del número 4 por teclado y luego presionar la tecla enter, se cierra la aplicación.

2. Desplegar delincuentes:

Una vez seleccionado desplegar delincuentes se despliega lo siguiente y el programa espera a la entrada de por teclado de la opción numérica deseada:

a. PreOrden:

Debe ingresar el número 1 por teclado y luego presionar la tecla enter. El programa mostrará por pantalla los datos de cada delincuente siguiendo el orden de nodo raíz, nodo izquierdo, nodo derecho.

b. InOrden:

Debe ingresar el número 2 por teclado y luego presionar la tecla enter. El programa mostrará por pantalla los datos de cada delincuente siguiendo el orden de nodo izquierdo, nodo raíz, nodo derecho.

c. PostOrden:

Debe ingresar el número 3 por teclado y luego presionar la tecla enter. El programa mostrará por pantalla los datos de cada delincuente siguiendo el orden de nodo izquierdo, nodo derecho, nodo raíz.

d. Por nivel:

Debe ingresar el número 4 por teclado y luego presionar la tecla enter. El programa mostrará por pantalla los datos de cada delincuente siguiendo el orden de izquierda a derecha por cada nivel del árbol.

Nota: Para mayor información acerca de los recorridos vea la sección de recorridos de árboles presente en este informe.

3. Modificar árboles:

Una vez seleccionado modificar árboles se despliega lo siguiente y el programa espera a la entrada de por teclado de la opción numérica deseada:

a. Agregar delincuente:

Debe ingresar el número 1 por teclado y luego presionar la tecla enter. Luego debe ingresar por teclado el rut del delincuente sin puntos ni dígito verificador, el nombre y apellido, el alias, la peligrosidad del delincuente y el delito. Luego de presionar enter el programa agrega al delincuente en los árboles ABB y AVL si este no existe ya en los mismos.

b. Capturar delincuente:

Debe ingresar el número 2 por teclado y luego presionar la tecla enter. Debe ingresar por teclado el rut del delincuente sin puntos o dígito verificador que quiera eliminar de ambos árboles.

4. Búsqueda árboles:

Una vez seleccionado búsqueda árboles se despliega lo siguiente y el programa espera a la entrada de por teclado de la opción numérica deseada:

a. Buscar por rut:

Debe ingresar el número 1 por teclado y luego presionar la tecla enter. Luego debe ingresar por teclado el rut del delincuente sin puntos ni dígito verificador que quiera buscar. Si el delincuente se encuentra registrado en los árboles, se procede a desplegar todos los datos de él, además se imprime el tiempo que tomó la búsqueda en ms.

b. Buscar por delito:

Debe ingresar el número 2 por teclado y luego presionar la tecla enter. Luego debe ingresar por teclado el tipo de delito que quiere buscar, los disponibles son: Hurto, Violación, Homicidio, Tráfico, Asalto, Vandalismo, o Fraude. Se despliega por pantalla todos los datos de todos los delincuentes que hayan cometido el delito ingresado, además se imprime el tiempo que tomó la búsqueda en ms.

Árboles utilizados

Árbol ABB:

Un ABB, es un árbol binario; ya sea vacío o en el cual cada nodo en el árbol contiene un identificador (dato) y se cumple:

A. Todos los identificadores en el subárbol izquierdo son menores (numérica o alfabéticamente) que el identificador en la raíz del nodo actual.
B. Todos los identificadores en el subárbol derecho son mayores que el identificador en la raíz del nodo actual.
C. Los subárboles izquierdo y derecho, son también ABB.

Métodos:

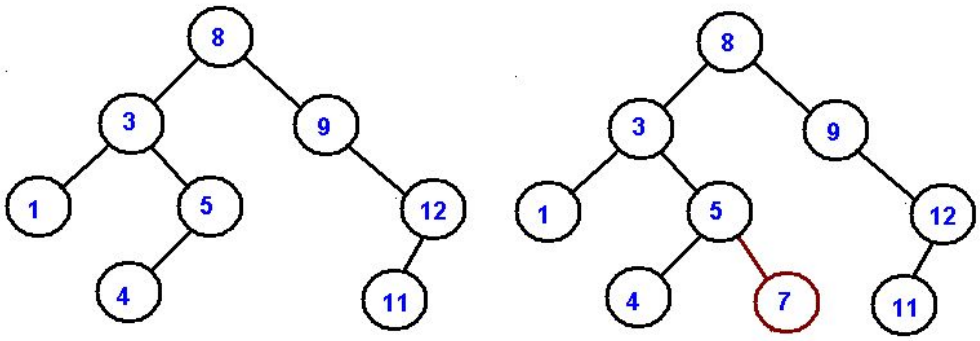
- Búsqueda

La búsqueda en un ABB siempre comienza desde la raíz. Se compara los datos almacenados en la raíz con la clave que estamos buscando. Si el nodo no contiene la clave, procedemos al hijo izquierdo o derecho según la comparación.

La búsqueda en un ABB tiene un tiempo O(h) de ejecución del peor caso, donde h es la altura del árbol. Como el árbol binario de búsqueda con n nodos tiene un mínimo de niveles O (log n), toma al menos O (log n) comparaciones para encontrar un nodo particular. Desafortunadamente, un árbol degenerado, puede reducir el tiempo de búsqueda a O (n), véase ventajas y desventajas para ver este caso.

- Añadir

El procedimiento de inserción es bastante similar a la búsqueda. Comenzamos en la raíz y bajamos recursivamente por el árbol buscando una ubicación en el ABB para insertar un nuevo nodo. Si el elemento que se va a insertar ya está en el árbol, se termina el método (no se inserta duplicados). El nuevo nodo siempre reemplazará una referencia NULL o nullptr.



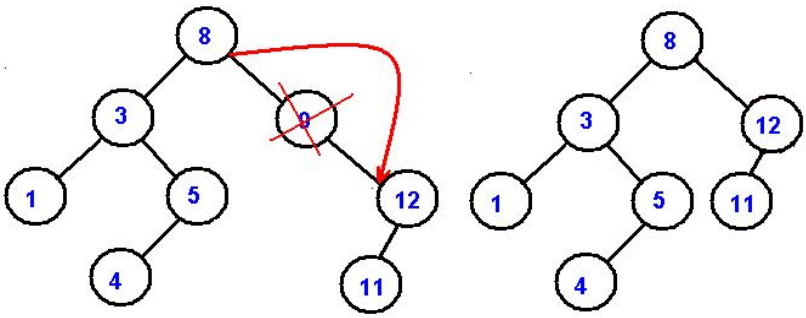
Antes de insertar 7 Después de insertar 7

❏ Eliminar

La eliminación es algo más complicada que la inserción. Hay varios casos que considerar. Un nodo que se eliminará puede:

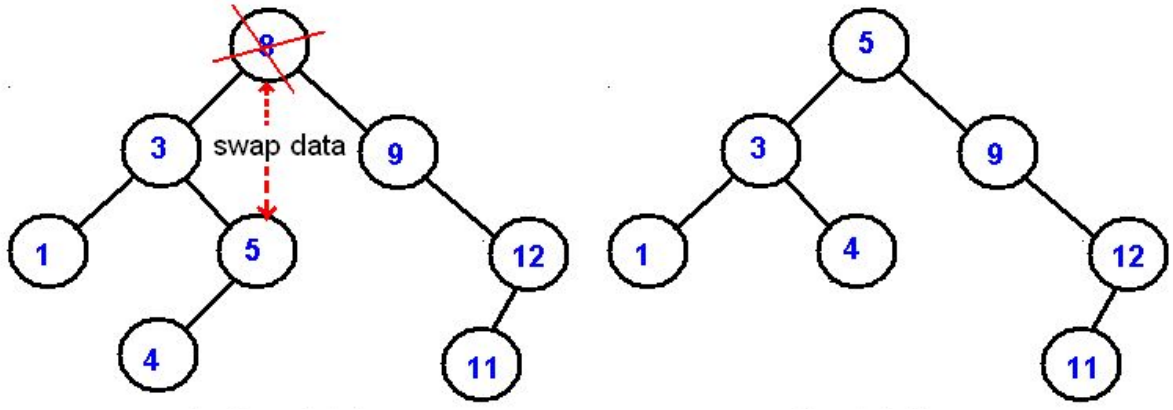
- No estar en el árbol.
- Es una hoja (No tiene hijos).
- Tiene solo un hijo.
- Tiene dos hijos.

Si no está en el árbol, no hay nada que hacer, si es una hoja se elimina ese nodo. Si el nodo a eliminar tiene solo un hijo, el procedimiento de eliminación es idéntico a eliminar un nodo de una lista vinculada; solo reemplazamos ese nodo por su hijo.



La eliminación de un nodo con dos hijos es menos directa. Si eliminamos dicho nodo, dividimos un árbol en dos subárboles y, por lo tanto, algunos hijos del nodo interno no serán accesibles después de la eliminación. En la imagen a continuación eliminamos el

8:



La estrategia de eliminación es la siguiente: Se reemplaza el nodo que se elimina con el nodo más grande en el subárbol izquierdo y luego elimine ese nodo más grande. Por simetría, el nodo que se elimina se puede intercambiar con el nodo más pequeño es el subárbol derecho.

En el mejor caso el tiempo es O(log_2 n) y en el peor caso es O(n).

Árbol AVL:

Es el primer árbol de búsqueda binario auto-balanceable que se ideó.

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. El factor de equilibrio puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles.

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.
La única diferencia de los nodos de un árbol AVL con los de un árbol binario común es la variable altura en la estructura nodo.

Métodos:

❏ Búsqueda

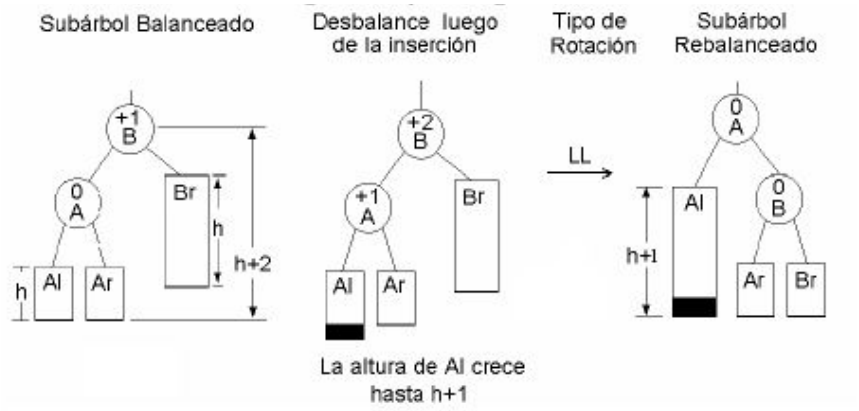
Para buscar un elemento es similar al ABB, con lo que en el mejor caso el tiempo es de O(log_2 n). En caso contrario, la búsqueda en el árbol AVL en el peor caso es de O(n).

❏ Añadir y eliminar

Similar al ABB con la gran diferencia que se tiene que verificar si está autobalanceado el AVL, de no estarlo se procede a realizar las rotaciones según sea el caso, existen 4 casos:

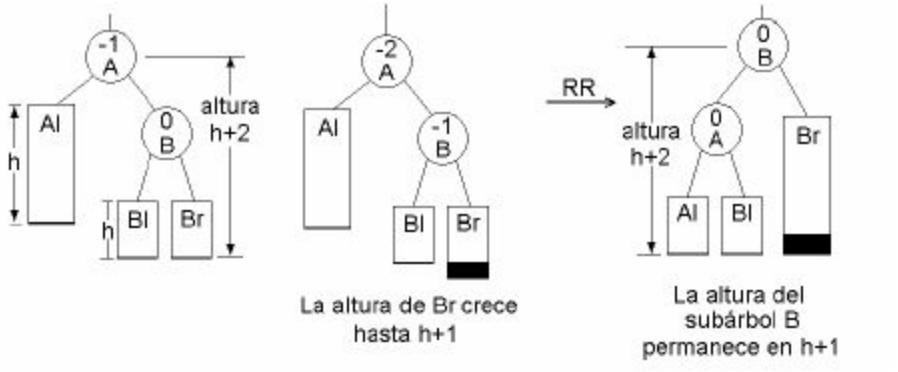
1. Rotación LL

   En rotación LL, cada nodo se mueve una posición hacia la izquierda desde la posición actual. Por ejemplo:



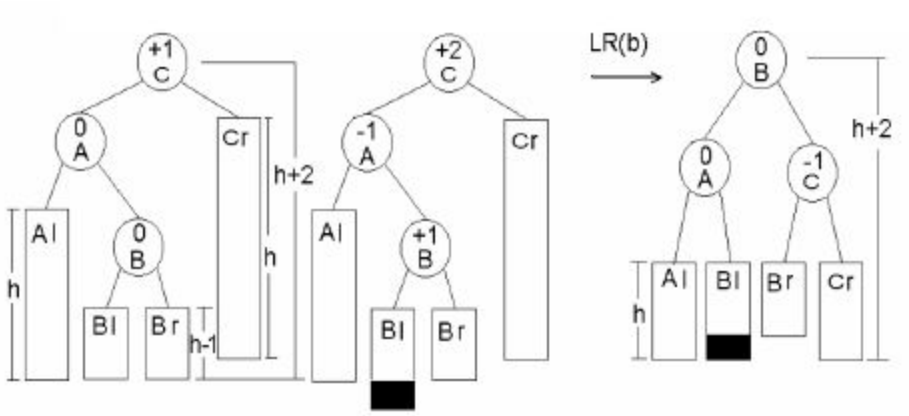
2. Rotación RR

   En la rotación RR, cada nodo se mueve una posición hacia la derecha desde la posición actual.



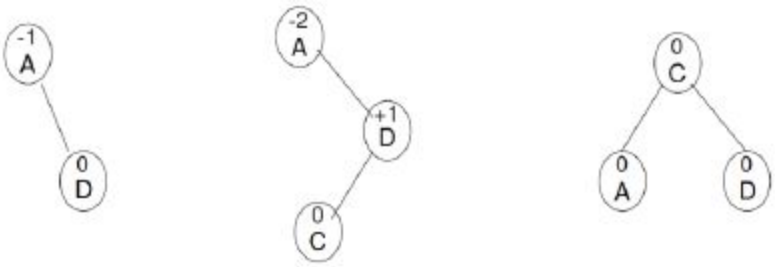
3. Rotación LR

En LR, primero cada nodo se mueve una posición hacia la izquierda y luego una posición hacia la derecha desde la posición actual (Se sube el mayor de los menores a partir del nodo con desbalance).

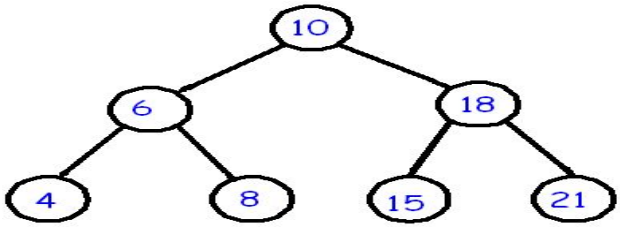


4. Rotación RL

En RL, primero cada nodo se mueve una posición hacia la derecha y luego una posición hacia la izquierda desde la posición actual (Se sube el menor de los mayores a partir del nodo con desbalance).



En ambos árboles el recorrido se realiza de las 4 siguientes maneras:



Tomando como ejemplo este árbol los recorridos serían:

1. **IN-ORDEN** .- Se utiliza el dato del puntero más a la izquierda, después el de la raíz y por último el de la derecha.
   Ejemplo: 4-6-8-10-15-18-21

2. **PRE-ORDEN** .- Se toma primero como dato la raíz, después el puntero más a la izquierda y por último a la derecha.
   Ejemplo: 10-6-4-8-18-15-21

3. **POST-ORDEN**.- Se desplaza al nodo más a la izquierda, después a la derecha y por último se utiliza el dato de la raíz.
   Ejemplo: 4-8-6-15-21-15-8

4. **POR NIVEL**.- Se desplaza de izquierda a derecha por cada nivel partiendo de la raíz del árbol.
   Ejemplo: 10-6-18-4-8-15-21

Donde el tiempo en cada recorrido es O(n) por qué se visita cada nodo del árbol para poder imprimir por pantalla.

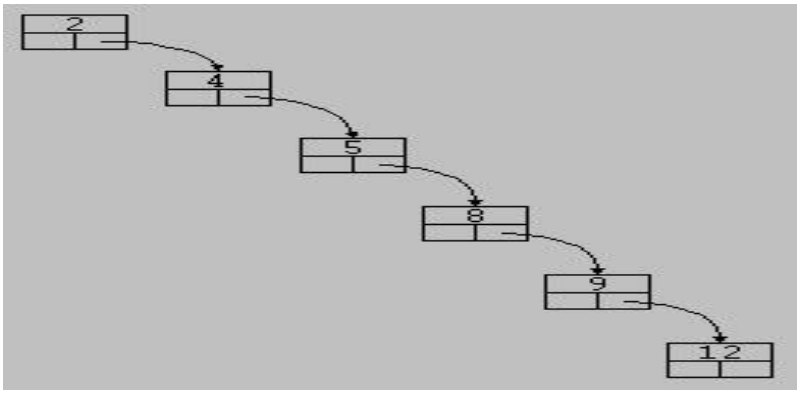
Ventajas y desventajas

El ABB tiene como ventaja:

- Refleja relaciones estructurales en los datos.
- Se usa para representar jerarquías.
- Proporciona una inserción y búsqueda eficiente, comparado a una lista dinámica ordenada.
- Los datos son muy flexibles, lo que permite mover los subárboles con un mínimo de esfuerzo.

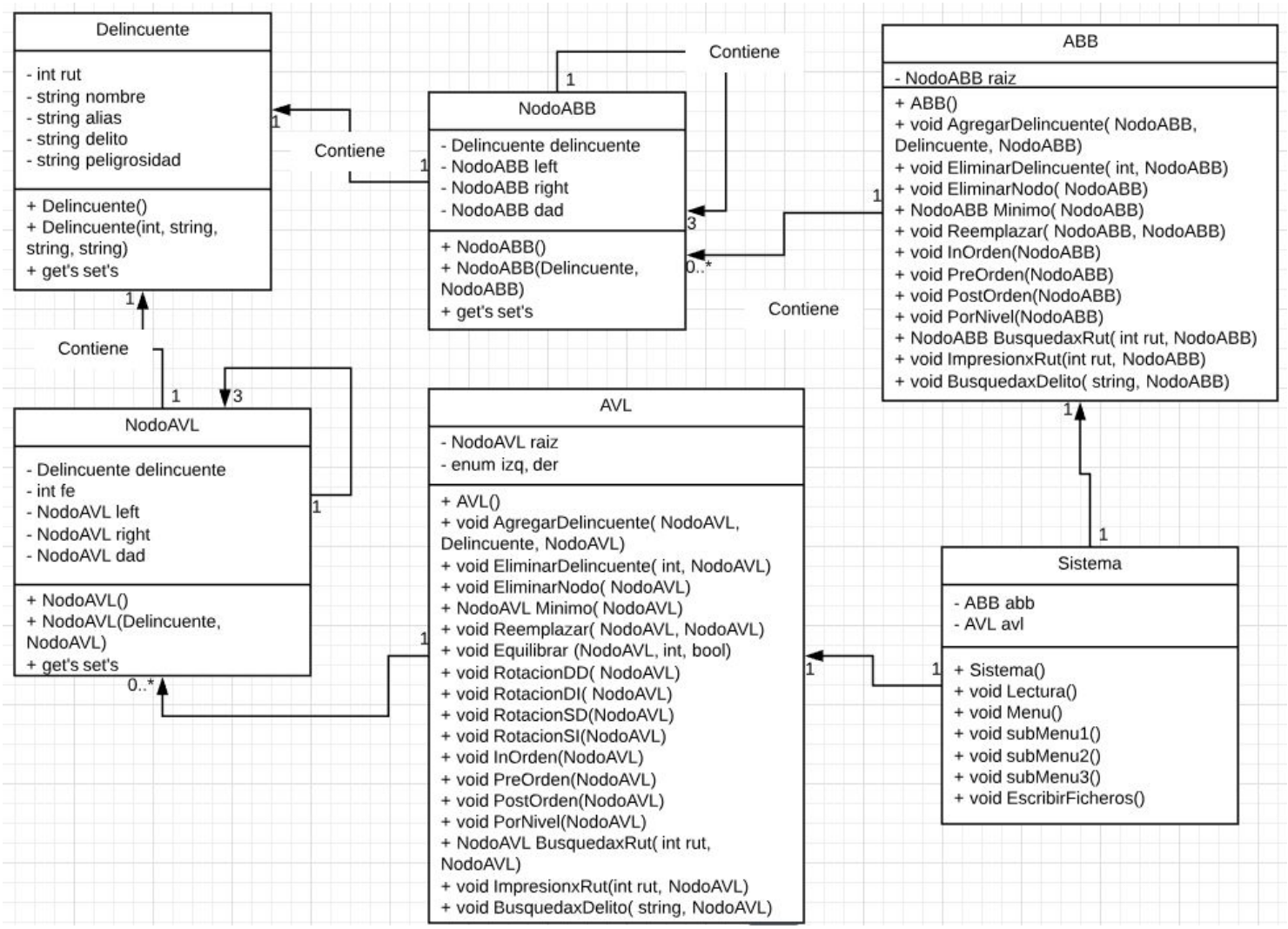
Por ejemplo en un árbol lleno que tenga n nodos el camino más largo que hay que recorrer es $log_2(n+1)$ comparado a una lista que es de n. Si n fuera 1023 en la lista se tiene que recorrer 1023 veces, en cambio con el ABB, $log_2(1024) = 10$ veces.

La gran desventaja del ABB es que puede ocurrir que sea un árbol degenerado según los datos ingresados, lo que conlleva a que se poco eficiente los procesos, véase la figura para entender más sobre este tema .



Este inconveniente se soluciona con él árbol AVL, ya que este se auto balancea a sí mismo, evitando casos como estos.

Diagrama de clases



Métodos utilizados

Clase	
Delincuente	- Delincuente(*parámetros) //constructor.*
	Get's y set's correspondientes a la clase para obtener los datos privados.

Clase	
NodoABB	- NodoABB(Delincuente, padre) *//constructor*
	Get's y set's correspondientes a la clase para obtener los datos privados.

Clase	
NodoAVL	- NodoAVL(Delincuente, padre) *//constructor*
	Get's y set's correspondientes a la clase para obtener los datos privados.

Clase	
ABB	- ABB() *//constructor*
	- AgregarDelincuente(NodoABB actual, Delincuente, NodoABB padre): Esta funcion recibe por parametro la raiz del arbol, el delincuente a agregarse, y el padre de la raíz(el primer nodo apunta a null) el recursivo va asignando el padre de dicho nodo.
	- EliminarDelincuente(int rut, NodoABB raíz): recibe por parámetros el rut a buscar para eliminar en árbol y la raíz del árbol, mediante recursividad busca dicho rut en todo el árbol. En caso de encontrar el rut envía el nodo de dicho rut a la función eliminarNodo.

	- EliminarNodo(NodoABB eliminado): función que recibe un nodo por parámetros y verifica los hijos que posee para efectuar la eliminación. En caso de poseer 2 hijos acude a la función mínimo para reasignar el valor a dicho nodo y por medio de recursividad eliminar el nodo duplicado, de lo contrario si es un nodo hoja o con 1 hijo acude a la función reemplazar y posteriormente elimina dicho nodo quitándole todos sus punteros.
	- Mínimo(NodoABB nodo): recibe por parámetros un nodo y busca por el árbol el valor más próximo por la derecha del árbol y retorna dicho nodo.
	- Reemplazar(NodoABB nodo, NodoABB nuevo): Función que asigna los punteros del nodo ingresado por parámetros y los ajusta al nodo nuevo para mantener la estructura del árbol.
	- InOrden(NodoABB raiz): funcion que recibe por parametro la raiz del arbol y recorre el arbol en in-orden.
	- PreOrden(NodoABB raiz): funcion que recibe por parametro la raiz del arbol y imprime el árbol en preorden
	- PostOrden(NodoABB raíz) función que recibe por parámetro la raíz del árbol y imprime el árbol en postorden
	- PorNivel(NodoABB raíz) función que recibe por parámetro la raíz del árbol y imprime el árbol en por nivel.
	- BusquedaxRut(int rut, NodoABB raíz): función que recibe por parámetro la raíz del árbol y realiza la búsqueda dependiendo si es mayor o menor el valor, retorna el nodo encontrado de acuerdo al rut recibido.
	- ImpresionxRut(int rut, NodoABB raíz): función que verifica si el nodo buscado por la función BusquedaxRut es un nodo válido y lo imprime por pantalla.
	- BusquedaxDelito(string delito, NodoABB raíz): función que recibe por parámetros el delito a buscar y realiza la busqueda desde la raiz del arbol imprimiendo todos los delincuentes que se emparejan con el delito ingresado.
	Get's y set's correspondientes a la clase para obtener los datos privados.

Clase	
AVL	- AVL() *//constructor*
	- AgregarDelincuente(NodoAVL raiz, Delincuente, NodoAVL padre, int rama): función que recibe por parámetros la raíz del árbol y de manera recursiva lo ubica en la posición que debe ir, posterior a eso realiza un llamado a la función equilibrar para verificar que el árbol está equilibrado.
	- EliminarDelincuente(int rut, NodoAVL raíz) recibe por parámetros el rut a buscar para eliminar en árbol y la raíz del árbol, mediante recursividad busca dicho rut en todo el árbol. En caso de encontrar el rut envía el nodo de dicho rut a la función eliminarNodo.
	- EliminarNodo(NodoAVL nodo) función que recibe un nodo por parámetros y verifica los hijos que posee para efectuar la eliminación. En caso de poseer 2 hijos acude a la función mínimo para reasignar el valor a dicho nodo y por medio de recursividad eliminar el nodo duplicado, de lo contrario si es un nodo hoja o con 1 hijo acude a la función reemplazar y posteriormente elimina dicho nodo quitándole todos sus punteros.
	- Mínimo(NodoAVL nodo) recibe por parámetros un nodo y busca por el árbol el valor más próximo por la derecha del árbol y retorna dicho nodo.
	- reemplazar(NodoAVL nodo, NodoAVL nuevo) Función que asigna los punteros del nodo ingresado por parámetros y los ajusta al nodo nuevo posterior a eso verifica que dicho nodo reemplazado siga manteniendo el equilibrio en el AVL.
	- Equilibrar(NodoAVL raiz, int rama, bool nuevo) función que recibe por parámetros el nodo por el cual comienza a reasignar los factores de equilibrio, recibe la rama correspondiente a 0 izquierda o 1 derecha, y recibe un booleano correspondiente a agregar un nuevo nodo = true o eliminar un nodo = false, una vez reasignado los FE verifica que dicho nodo se encuentra equilibrado, de encontrarse en un estado de desequilibrio acude a las rotaciones.
	- RotacionDD(NodoAVL nodo) corresponde a la rotación de doble derecha o rotación left right.
	- RotacionDI(NodoAVL nodo) corresponde a la rotación de doble izquierda o rotación right left

	- RotacionSD(NodoAVL nodo) corresponde a la rotación de simple derecha o right right
	- RotacionSI(NodoAVL nodo) corresponde a la rotación de simple izquierda o left left
	- InOrden(NodoAVL raiz): funcion que recibe por parametro la raiz del arbol y recorre el arbol en in-orden.
	- PreOrden(NodoAVL raiz): funcion que recibe por parametro la raiz del arbol y imprime el árbol en preorden
	- PosOrden(NodoAVL raíz) funcion que recibe por parámetro la raiz del arbol y imprime el arbol en post-orden
	- PorNivel(NodoAVL raíz) función que recibe por parámetro la raíz del árbol y imprime el árbol en por nivel.
	- BusquedaxRut(int rut, NodoAVL raíz): función que recibe por parámetro la raíz del árbol y realiza la búsqueda dependiendo si es mayor o menor el valor, retorna el nodo encontrado de acuerdo al rut recibido.
	- ImpresionxRut(int rut, NodoAVL raíz): función que verifica si el nodo buscado por la función BusquedaxRut es un nodo válido y lo imprime por pantalla.
	- BusquedaxDelito(string delito, NodoAVL raíz): función que recibe por parámetros el delito a buscar y realiza la busqueda desde la raiz del arbol imprimiendo todos los delincuentes que se emparejan con el delito ingresado.

Clase	
Sistema	- Lectura() Función que permite leer el archivo datos.txt y agregarlos a los árboles correspondiente.
	- Menú() Funcion de creacion menú interactivo para menú principal
	- subMenu1() Funcion de creacion menú interactivo para el submenú mostrar información
	- subMenu2() Funcion de creacion menú interactivo para submenú modificar árboles

	- subMenu3() Funcion de creacion menu interactivo para subMenu busqueda en arboles.
	- EscribirFicheros() función que permite al terminar el programa, agregar los datos al archivo datos.txt
	Get's y set's correspondientes a la clase para obtener los datos privados.

Horas en Trabajo

Fecha	Avances	Encargado
21-24 de Mayo	- Confección de diagrama de clases	- Yeison Olivares
23 de Mayo - 2 de Junio	- Realización del código del programa	- Ambos
23-30 de Mayo 23-31 de Mayo	- Arbol ABB - Arbol AVL	- Yeison Olivares - Rodrigo Dominguez
30 de Mayo - 2 de Junio	- Comentarios en código - Arreglar errores o sucesos inesperados	- Ambos - Ambos

Conclusión

Mediante la aplicación de lo aprendido en clases, se confeccionó este informe para tener una documentación del programa y del funcionamiento del código, donde cada punto aporta comprensión tanto para el desarrollador como el usuario final, estos se especifican en:

1) Diagrama de clases:
    a) Diagrama visual en el cual se expone de manera sencilla al usuario final, en este caso cualquier usuario en especial Jimmy que utilice la aplicación, sobre la relación de los distintos objetos del programa.
2) Métodos Usados:
    a) Breve explicación de cada clase indicando sus parámetros y métodos.

3) Manual de usuario:
    a) Explicación en lenguaje “cotidiano” para el usuario final, que utilizará la aplicación.