RENTA DE ACTIVOS

MANUAL TECNICO

Presentación

El siguiente manual ha sido desarrollado con la finalidad de dar a conocer la información necesaria para realizar el mantenimiento, instalación y exploración del software de **Renta de Activos**, el cual consta de diferentes funcionalidades. El manual ofrece la información necesaria de ¿Cómo está desarrollado el software? Para que la persona (Desarrollador en C++) que quiera editar el software lo haga de una manera apropiada, dando a conocer la estructura del desarrollo del aplicativo.

Resumen

El manual detalla los aspectos técnicos e informativos del software de **Renta de Activos** con la finalidad de explicar la estructura del aplicativo al personal que quiera administrarlo, editarlo o configurarlo. La siguiente guía se encuentra dividida en las herramientas que se usaron para el desarrollo del software con una breve explicación paso a paso. El aplicativo maneja diferentes funcionalidades los cuales se explicara que funcionamiento realiza cada uno de ellos, dando sugerencias para el debido uso del sistema informático.

Introducción

El manual se realiza con el fin de detallar el software para la **Renta de Activos** en términos técnicos para aquella persona que vaya a administrar, editar o configurar el aplicativo lo haga de una manera apropiada. El documento se encuentra dividido en las siguientes secciones:

- ENTORNO DE DESARROLLO: Se darán a conocer las herramientas de desarrollo que se utilizaron para el software, así como se darán breves explicaciones de instalaciones y configuraciones necesarias de las herramientas anteriormente expuestas
- DESCRIPCION DE CODIGO FUENTE: Se describirán las clases y métodos que fueron implementados dentro del código fuente de la aplicación
- DESCRIPCION DE TDAs: Se explicaran las estructuras de datos que se utilizaron para el desarrollo de la aplicación

ENTORNO DE DESARROLLO

El aplicativo de **Renta de Activos** tiene la finalidad de administrar las Rentas de Activos que pertenezcan a determinados Clientes. Se recomienda que el siguiente manual sea manipulado únicamente por la persona que quiera administrar, editar o configurar el software para la **Renta de Activos** para velar por la seguridad de los datos que se almacenan. A continuación se presenta la instalación, configuración y descripción de las herramientas utilizadas para el desarrollo de la aplicación:

WSL (Windows Subsistem Linux)

- Descripción

Subsistema de Windows para Linux (WSL) es una característica de Windows que permite ejecutar un entorno Linux en la máquina de Windows, sin necesidad de una máquina virtual independiente ni de arranque dual. WSL está diseñado para proporcionar una experiencia perfecta y productiva para los desarrolladores que quieren usar Windows y Linux al mismo tiempo.

WSL 2 es el tipo de distribución predeterminada al instalar una distribución de Linux. WSL 2 usa tecnología de virtualización ligera. Las distribuciones de Linux se ejecutan como contenedores aislados dentro de la máquina virtual administradas de WSL 2. WSL 2 aumenta el rendimiento del sistema de archivos y agrega compatibilidad completa de llamadas del sistema en comparación con la arquitectura WSL 1.

- Instalación

Si tienes Linux o macOS no es necesario la instalación de WSL. Para la instalación de WSL en Windows desde la página oficial de Microsoft dirígete al siguiente enlace: https://learn.microsoft.com/es-es/windows/wsl/install en donde tendrás la guía completa de como descargar e instalar WSL en tu equipo de Windows.

- Configuración

Después de haber instalado WSL se tendrán que ejecutar los siguientes comandos:

sudo apt update: Con este comando se buscaran actualizaciones para los paquetes de la distribución de Linux instalada

sudo apt upgrade: Con este comando se aplicaran las actualizaciones (si los hay) de los paquetes que anteriormente se encontraron. Si en el comando anterior no se encontraron paquetes por actualizar no es necesario ejecutar este comando.

sudo apt install cmake gcc clang gdb build-essential: Con este comando se instalaran las herramientas, administradores, compiladores y debugers necesarios para el manejo de C++

CLion

- Descripción

IDE que viene de la mano de JetBrains una empresa de desarrollo de software bastante conocida por la creación de varias herramientas y lenguajes de programación Kotlin. CLion es un IDE enfocado para el desarrollo en los lenguajes de programación C y C++, CLion es un IDE multiplataforma por lo que puede ser utilizado en Linux, macOS y Windows integrado con el sistema de compilación CMake.

Además de C y C++, CLion admite otros lenguajes directamente o mediante complementos: Kotlin, Python, Rust, Swift y otros. CLion al igual que muchos IDE cuanta con la función de completar el código fácilmente, con lo cual puede ayudarte a ahorrar bastante tiempo en completar las sintaxis de tu código que estés escribiendo en él.

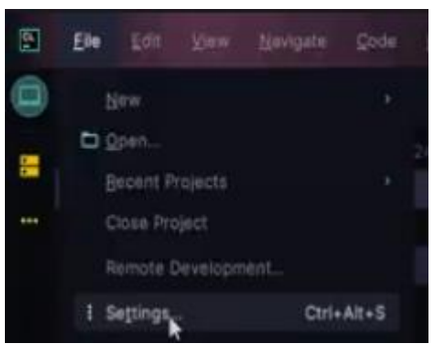
- Instalación

Para la instalación de CLion desde la página oficial de JetBrains dirígete al siguiente enlace: https://www.jetbrains.com/es-es/clion/ en donde podrás descargar CLion para su posterior instalación.

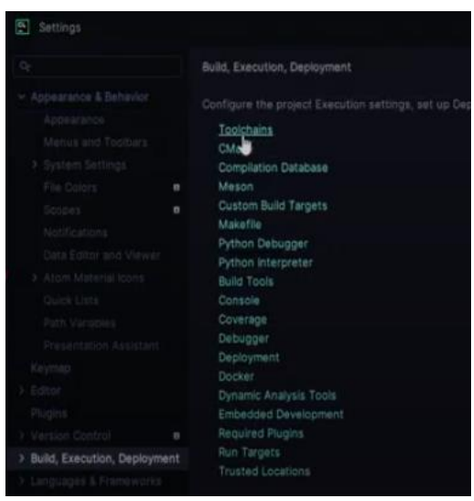
- Configuración

Si te encuentras en Linux o macOS, no es necesario hacer esta configuración. Si te encuentras en Windows debes hacer la siguiente configuración a CLion.

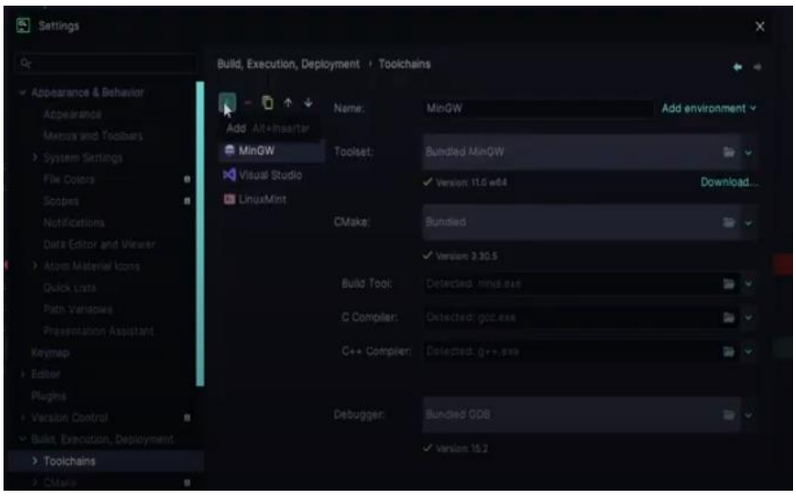
Dirígete a: Settings…:



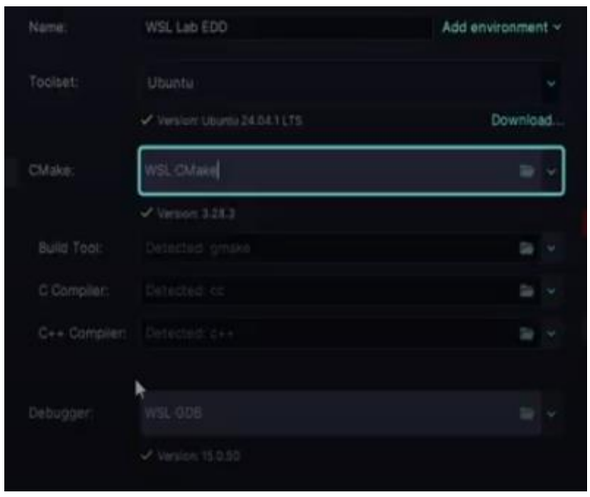
Dirígete a: Build, Execution, Deployment -> Toolchains



Luego en Add -> WSL:



De las siguientes configuraciones la más importante en tener es la de Toolset, en donde debes seleccionar la distribución de Linux que instalaste en WSL



Graphviz

- Descripción

Graphviz es un software de visualización de gráficos de código abierto. La visualización de gráficos es una forma de representar información estructural como diagramas de redes y gráficos abstractos. Tiene importantes aplicaciones en redes, bioinformática, ingeniería de software, diseño de base de datos y sitios web, aprendizaje automático y en interfaces visuales para otros dominios técnicos.

Los programas de diseño de Graphviz toman descripciones de gráficos en un lenguaje de texto simple y crean diagramas en formato útiles, como imágenes y SVG para páginas web; PDF Postscript para incluir en otros documentos; o visualizar en un navegador concreto, como opción de colores, fuentes, diseños de nodos tabulares, estilos de líneas, hipervínculos y formas personalizadas.

- Instalación

Para los que instalaron WSL, es recomendable que instalen Graphviz en los dos sistemas operativos por si acaso uno de los dos fallara (no debería) tener el otro de reserva, de igual forma este software es ligero en descarga e instalación.

Linux: ejecutar los siguientes comandos en la consola/terminal de WSL o Linux directamente.

sudo apt update: Con este comando se buscaran actualizaciones para los paquetes de la distribución de Linux instalada

sudo apt upgrade: Con este comando se aplicaran las actualizaciones (si los hay) de los paquetes que anteriormente se encontraron. Si en el comando anterior no se encontraron paquetes por actualizar no es necesario ejecutar este comando.

sudo apt install graphviz: Este comando instalara todo lo necesario para tener Graphviz

Windows: Para la instalación de Graphviz desde su página oficial dirígete al siguiente enlace: https://graphviz.org/download/ en donde podrás descargar el ejecutable para Windows y así hacer su instalación.

DESCRIPCION DE CODIGO FUENTE

- **main.cpp:** Es la Clase principal en donde se ejecuta la aplicación
    - main(): Es el Metodo principal en donde se ejecuta toda la aplicación:

```
int main() {
    auto *menuInicio = new Menus();
    menuInicio->cargarDatos();
    menuInicio->menuPrincipal();

    return 0;
}
```

- **Utils.cpp:** Es la Clase que se encarga de tener métodos estáticos que son de uso general para toda la ejecución del programa
    - generarIDAlfanumerico(): Es el Método que se encarga de generar un ID único alfanumérico que servirá para identificar a los diferentes activos que se vallan registrando en la aplicación, así como también generar los identificadores de las transacciones que se realicen entre los usuario.

```
std::string Utils::generarIDAlfanumerico() {
    const int longitudID = 15;
    const std::string alfanumericos = "abcdefghijklmnopqrstuvwxyz0123456789";
    std::string ID;
    for (int i = 0; i < longitudID; i++) {
        ID += alfanumericos[rand()%(alfanumericos.size()-1)];
    }
    return ID;
}
```

    - isEquals(string, string): Este Método se encarga de evaluar si dos cadenas de son iguales sin importar mayúsculas y minúsculas, es utilizado para cuando se reciben nombres de departamentos, empresas o usernames, y de esa forma verificar que sean los datos que se esperan

```
bool Utils::isEquals(std::string cadena1, std::string cadena2) {
    int longitud1 = cadena1.length();
    int longitud2 = cadena2.length();

    if (longitud1 != longitud2) return false;

    for (int i = 0; i < longitud1; i++) {
        if (std::tolower(static_cast<unsigned char>(cadena1[i]))
            != std::tolower(static_cast<unsigned char>(cadena2[i]))) {
            return false;
        }
    }
    return true;
}
```

- verificarEntradaNumerica(int, string): Método que verifica si el dato ingresado del usuario por consola es válido para considerarse numérico, si el dato corresponde a un número entonces se sigue con el flujo normal del programa, de lo contrario se le informara que ha cometido un error y debe ingresar un valor numérico, esto se repetirá hasta que el usuario ingrese un valor numérico valido.

```cpp
void Utils::verificarEntradaNumerica(int &valorAsignado, const std::string textoMostrar) {
    std::string entrada;
    while (true) {
        try {
            std::cout << textoMostrar;
            std::getline( [&] std::cin,  [&] entrada);
            std::stringstream ss(entrada);
            if (!(ss >> valorAsignado) || !(ss.eof())) {
                throw std::runtime_error("Debe Ingresar un Valor Numerico");
            }
            break;
        } catch (const std::exception &e) {
            std::cout <<">> Error!!!: " << e.what() << std::endl;
        }
    }
}
```

- **Menus.cpp:** Clase que define los diferentes menús que se tendrán durante la aplicación, esta es la clase más visual con respecto al usuario que se tiene.
    - cargarDatos(): Método que realiza una carga de datos previa a la inicialización de la aplicación, esto con el fin de poder tener una base de datos dentro de la aplicación

```cpp
void Menus::cargarDatos() {
    std::cout << "RECUPERANDO DATOS..." << std::endl;

    auto *usuario1 = new Usuario( nombre: "Elian Estrada", username: "elian_estrada", password: "1234");
    matrizDispersa->insretarUsuario(usuario1, departamento: "guatemala", empresa: "igss");
    usuario1->getArbol()->insertar(new Activo( nombre: "madera", descripcion: "madera para albañil", diasDisponibles: 20));
    usuario1->getArbol()->insertar(new Activo( nombre: "martillos", descripcion: "martillos para madera", diasDisponibles: 10));
    usuario1->getArbol()->insertar(new Activo( nombre: "caladora", descripcion: "caladora para cortar maderas prefabricadas", diasDisponibles: 15));
    usuario1->getArbol()->insertar(new Activo( nombre: "barreno", descripcion: "barreno para concreto", diasDisponibles: 5));

    matrizDispersa->insretarUsuario(new Usuario( nombre: "Juan Perez", username: "juanito", password: "4567"), departamento: "jutiapa", empresa: "max");
    matrizDispersa->insretarUsuario(new Usuario( nombre: "Pedro Rodriguez", username: "pedrito", password: "48956"), departamento: "jalapa", empresa: "usac");

    auto *usuario4 = new Usuario( nombre: "Maria Fernanda", username: "mafer", password: "54312");
    matrizDispersa->insretarUsuario(usuario4, departamento: "peten", empresa: "cinepolis");
    usuario4->getArbol()->insertar(new Activo( nombre: "audifonos", descripcion: "audifonos para grabaciones de estudio", diasDisponibles: 10));
    usuario4->getArbol()->insertar(new Activo( nombre: "microfonos", descripcion: "microfonos de todo tipo", diasDisponibles: 8));
    usuario4->getArbol()->insertar(new Activo( nombre: "pedestales", descripcion: "pedestales para microfonos grandes y pequeños", diasDisponibles: 12));
    usuario4->getArbol()->insertar(new Activo( nombre: "atriles", descripcion: "atriles para colocar ojas con gancho", diasDisponibles: 14));

    matrizDispersa->insretarUsuario(new Usuario( nombre: "Juan Manuel", username: "juanma", password: "32897"), departamento: "guatemala", empresa: "usac");

    auto *usuario6 = new Usuario( nombre: "Carlos Perez", username: "casimiro", password: "721896");
    matrizDispersa->insretarUsuario(usuario6, departamento: "guatemala", empresa: "max");
    usuario6->getArbol()->insertar(new Activo( nombre: "balanza", descripcion: "balanza con maximo de 25kg", diasDisponibles: 15));
    usuario6->getArbol()->insertar(new Activo( nombre: "canastas", descripcion: "canastas para frutas y verduras", diasDisponibles: 45));
    usuario6->getArbol()->insertar(new Activo( nombre: "linternas", descripcion: "linternas para alumbrar cuartos oscuros", diasDisponibles: 10));
    usuario6->getArbol()->insertar(new Activo( nombre: "cargadores", descripcion: "cargadores de telefonos tipo c", diasDisponibles: 5));
    usuario6->getArbol()->insertar(new Activo( nombre: "cables", descripcion: "cables de todo tipo", diasDisponibles: 10));
    usuario6->getArbol()->insertar(new Activo( nombre: "lazos", descripcion: "lazos para tender ropa", diasDisponibles: 20));

    auto *usuario7 = new Usuario( nombre: "Fernando Mendez", username: "fuego03", password: "891346");
    matrizDispersa->insretarUsuario(usuario7, departamento: "jutiapa", empresa: "cinepolis");
    usuario7->getArbol()->insertar(new Activo( nombre: "termos", descripcion: "pequeños termos para bebidas calientes", diasDisponibles: 10));
    usuario7->getArbol()->insertar(new Activo( nombre: "maletas", descripcion: "maletas desde pequeñas a grandes", diasDisponibles: 15));
    usuario7->getArbol()->insertar(new Activo( nombre: "peliculas", descripcion: "todo tipo de peliculas de accion", diasDisponibles: 5));

    matrizDispersa->insretarUsuario(new Usuario( nombre: "Alejandra Guzman", username: "azurdia", password: "780145"), departamento: "jutiapa", empresa: "usac");
```

- menuPrincipal(): Este método genera la primer interacción visual entre el usuario y la aplicación, en donde se podrá ingresar directamente a la aplicación pasando por un sistema de login, o simplemente con la opción de salir de la aplicación.

```
void Menus::menuPrincipal() {
    int opcionElegida;
    do {
        std::cout << "\n>> %%%%%%%%%%%%%%%%%%%% Bienvenido a \"Renta de Activos\" %%%%%%%%%%%%%%%%%%%%" <<std::endl;
        std::cout << ">> %%%%%%%%%%%%%%%%%%%%          Menu Principal           %%%%%%%%%%%%%%%%%%%%" <<std::endl;
        std::cout << ">> %%%%%%%%%%%%%%%%%%%%          1.Ingresar               %%%%%%%%%%%%%%%%%%%%" <<std::endl;
        std::cout << ">> %%%%%%%%%%%%%%%%%%%%          2.Salir                  %%%%%%%%%%%%%%%%%%%%" <<std::endl;
        Utils::verificarEntradaNumerica( [&] opcionElegida, textoMostrar: [] ">> Opcion: ");
        switch (opcionElegida) {
            case 1:
                login();
                break;
            case 2:
                std::cout << ">> Saliendo..." << std::endl;
                break;
            default:
                std::cout << ">> Opcion ingresada invalida. Intente otra vez" << std::endl;
                break;
        }
    } while (opcionElegida != 2);
}
```

- login():Método en donde se ejecuta el inicio de sesión de los usuarios registrados dentro de la aplicación, siempre que las credenciales de usuario coincidan con algún usuario ya registrado se seguirá con el flujo normal del programa, de lo contrario se mostrara un mensaje informando que el usuario no está registrado y se regresara al menú anterior.

```
void Menus::login() {
    std::cout << "\n>> %%%%%%%%%%%%%%%%%%%% Login %%%%%%%%%%%%%%%%%%%%" <<std::endl;

    std::string username;
    std::string password;
    std::cout << ">> ... Ingrese su Usuario ..." <<std::endl;
    std::getline( [&] std::cin, [&] username);
    std::cout << ">> ... Ingrese su Password ..." <<std::endl;
    std::getline( [&] std::cin, [&] password);

    if (Utils::isEquals( [] username, [] USERNAME_ADMIN) && Utils::isEquals( [] password, [] PASSWORD_ADMIN)) {
        menuAdministrador();
    } else {
        std::string departamento;
        std::string empresa;
        std::cout << ">> ... Ingrese su Departamento ..." <<std::endl;
        std::getline( [&] std::cin, [&] departamento);
        std::cout << ">> ... Ingrese su Empresa ..." <<std::endl;
        std::getline( [&] std::cin, [&] empresa);

        NodoMatriz *aux = matrizDispersa->existeNodoInterseccion( [] departamento, [] empresa);
        if (aux == nullptr) {
            std::cout << ">> Error!!!. El Usuario Ingresado NO esta Registrado" << std::endl;
        } else {
            do {
                if (Utils::isEquals(aux->getUsuario()->getUsername(), [] username) && Utils::isEquals(aux->getUsuario()->getPassword(), [] password)) {
                    NodoMatriz *usuarioLogeado = aux;
                    menuUsuario(usuarioLogeado);
                    return;
                }
                aux = aux->getAtras();
            } while (aux != nullptr);
            std::cout << ">> Error!!!. El Usuario Ingresado NO esta Registrado" << std::endl;
        }
    }
}
```

- menuAdministrador(): Este Método genera el menú que tendrá el usuario con rol de administrador dentro del programa, se muestran las diferentes opciones a realizar que tiene el usuario administrador dentro del programa, en donde podrá navegar dentro del mismo.

- registroUsuario(): Método en donde se lleva a cabo el registro de un nuevo usuario dentro de la aplicación, para ello se solicitaran los datos necesarios para el registro que se lleva a cabo al final.

```
void Menus::registroUsuario() {
    std::cout << "\n>> %%%%%%%%%%%%%%%%%%%% Registro Usuario %%%%%%%%%%%%%%%%%%%%" <<std::endl;
    std::cout << ">> ... Ingresar UserName Usuario...:" <<std::endl;
    std::string userName;
    std::getline( [&] std::cin, [&] userName);

    std::cout << ">> ... Ingresar Password...:" <<std::endl;
    std::string password;
    std::getline( [&] std::cin, [&] password);

    std::cout << ">> ... Ingresar Departamento...:" <<std::endl;
    std::string departamento;
    std::getline( [&] std::cin, [&] departamento);

    std::cout << ">> ... Ingresar Empresa...:" <<std::endl;
    std::string empresa;
    std::getline( [&] std::cin, [&] empresa);

    std::cout << ">> ... Ingresar Nombre Completo...:" <<std::endl;
    std::string nombreCompleto;
    std::getline( [&] std::cin, [&] nombreCompleto);

    matrizDispersa->insretarUsuario(new Usuario( nombreCompleto, userName, password), departamento, empresa);
}
```

- reporteActivosUsuario(): Método que genera el Reporte de Activos de un Determinado Usuario con sus Credenciales, siempre que se encuentre al usuario con las credenciales encontradas se generara el reporte de lo contrario se mostrara un mensaje informando que no se ha encontrado al usuario solicitado.

```
void Menus::reporteActivosUsuario() {
    std::cout << "\n>> %%%%%%%%%%%%%%%%%%%% Activos de Usuario %%%%%%%%%%%%%%%%%%%%" <<std::endl;
    std::cout << ">> ... Ingresar UserName del Usuario...:" <<std::endl;
    std::string userName;
    std::getline( [&] std::cin, [&] userName);

    std::cout << ">> ... Ingresar el Departamento...:" <<std::endl;
    std::string departamento;
    std::getline( [&] std::cin, [&] departamento);

    std::cout << ">> ... Ingresar la Empresa...:" <<std::endl;
    std::string empresa;
    std::getline( [&] std::cin, [&] empresa);

    NodoMatriz *aux = matrizDispersa->existeNodoInterseccion( departamento, empresa);
    if (aux != nullptr) {
        do {
            if (Utils::isEquals(aux->getUsuario()->getUsername(), userName)) {
                aux->getUsuario()->getReporteArbol()->reporteActivosUsuario( nodo: aux->getUsuario()->getArbol()->getRaiz(), userName);
                break;
            }
            aux = aux->getAtras();
        } while (aux != nullptr);

        if (aux == nullptr) std::cout << ">> No se Encontro al Usuario con las Credenciales Ingresadas" << std::endl;

    } else {
        std::cout << ">> No se Encontro al Usuario con las Credenciales Ingresadas" << std::endl;
    }
}
```

- menuUsuario(*usuarioLogeado): Este Método genera el menú que tendrán los diferentes usuarioregistrados dentro del programa, se muestran las diferentes opciones a realizar que tiene el usuario dentro del programa, en donde podrá navegar dentro del mismo.

- modificarActivo(*usuarioLogeado): Método que se encarga de mostrar los Activos que el Usuario actual tiene disponibles para su modificación, en donde si quiere hacer alguna modificación deberá de ingresar el ID del Activo como también ingresar la nueva descripción que tendrá el Activo.

```
void Menus::modificarActivo(NodoMatriz *usuarioLogeado) {
    std::cout << "\n>> %%%%%%%%%%%%%%%%%%%% Modificar Activo %%%%%%%%%%%%%%%%%%%%" <<std::endl;
    if (!usuarioLogeado->getUsuario()->getArbol()->mostrarActivos(true)) {
        std::cout << ">> No hay Activos por Mostrar..." <<std::endl;
        return;
    }
    std::cout << ">> ...Ingresar ID de Activo a Modificar...";
    std::string idActivo;
    std::getline( [&] std::cin, [&] idActivo);

    std::cout << ">> ...Ingresar Descripcion Nueva...";
    std::string nuevaDescripcion;
    std::getline( [&] std::cin, [&] nuevaDescripcion);

    usuarioLogeado->getUsuario()->getArbol()->modificarActivo( idActivo, nuevaDescripcion);
    regresarMenu();
}
```

- regresarMenu(): Método que se encarga de mostrar un mensaje para que el usuario pueda regresar al menú anterior en donde se encontraba

```
void Menus::regresarMenu() {
    int valorSalida;
    do {
        Utils::verificarEntradaNumerica( [&] valorSalida, textoMostrar: ">> ...Ingresar 1 para Regresar al Menu...: ");
    } while (valorSalida != 1);
    std::cout << ">> Regresando..." << std::endl;
}
```

- **Usuario.h:** Es la definición de la Clase Usuario en donde no tiene nada de especial con respecto a los métodos, pero que es la representación de un usuario dentro de la aplicación.

```
class Usuario {

private:
    std::string nombre;
    std::string username;
    std::string password;
    ArbolAVL *arbol;
    ReporteArbolAVL *reporte;

public:
    Usuario(std::string nombre, std::string username, std::string password);

    std::string getNombre();
    void setNombre(std::string nombre);
    std::string getUsername();
    void setUsername(std::string username);
    std::string getPassword();
    void setPassword(std::string password);
    ArbolAVL *getArbol();
    void setArbol(ArbolAVL *arbol);
    ReporteArbolAVL *getReporteArbol();
    void setReporte(ReporteArbolAVL *reporte);

};
```

- **Activo.h:** Es la definición de la Clase Activo en donde no tiene nada de especial con respecto a los métodos, pero que es la representación de un activo dentro de la aplicación.

```
class Activo {

private:
    std::string id;
    std::string nombre;
    std::string descripcion;
    bool disponibilidad;
    int diasDisponibles;

public:
    Activo(std::string nombre, std::string descripcion, int diasDisponibles);

    std::string getId();
    void setId(std::string id);

    std::string getNombre();
    void setNombre(std::string nombre);

    std::string getDescripcion();
    void setDescripcion(std::string descripcion);

    bool getDisponibilidad();
    void setDisponibilidad(bool disponibilidad);

    int getDiasDisponibles();
    void setDiasDisponibles(int diasDisponibles);

};
```

- **Transaccion.h:** Es la definición de la Clase Transacción en donde no tiene nada de especial con respecto a los métodos, pero que es la representación de una transacción dentro de la aplicación.

```
class Transaccion {

private:
    std::string id;
    Activo *activo;
    std::string usuario;
    std::string departamentoUsuario;
    std::string empresaUsuario;
    std::string fecha;
    int dias;

public:
    Transaccion(Activo *activo, std::string usuario, std::string departamentoUsuario, std::string empresaUsuario, std::string fecha, int dias);

    std::string getId();
    void setId(std::string id);
    Activo *getActivo();
    void setActivo(Activo *activo);
    std::string getUsuario();
    void setUsuario(std::string usuario);
    std::string getDepartamentoUsuario();
    void setDepartamentoUsuario(std::string nombreDepartamento);
    std::string getEmpresaUsuario();
    void setEmpresaUsuario(std::string nombreEmpresa);
    std::string getFecha();
    void setFecha(std::string fecha);
    int getDias();
    void setDias(int dias);

};
```

- **NodoArbol.h:** Es la definición de la Clase NodoArbol en donde no tiene nada de especial con respecto a los métodos, pero que es la representación de un nodo dentro de la estructura de Árbol AVL.

```
class NodoArbol {

private:
    Activo *activo;
    NodoArbol *hijoIzquierdo;
    NodoArbol *hijoDerecho;
    int factorEquilibrio;

public:
    NodoArbol(Activo *activo);

    Activo *getActivo();
    void setActivo(Activo *activo);

    NodoArbol *&getHijoIzquierdo();
    void setHijoIzquierdo(NodoArbol *hijoIzquierdo);

    NodoArbol *&getHijoDerecho();
    void setHijoDerecho(NodoArbol *hijoDerecho);

    int getFactorEquilibrio();
    void setFactorEquilibrio(int factorEquilibrio);

};
```

- **NodoLista.h:** Es la definición de la Clase NodoLista en donde no tiene nada de especial con respecto a los métodos, pero que es la representación de un nodo dentro de la estructura de Lista Circular Doblemente Enlazada.

```
class NodoLista {

private:
    NodoLista *siguiente;
    NodoLista *anterior;
    Transaccion *transaccion;

public:
    NodoLista(Transaccion *transaccion);

    NodoLista *getSiguiente();
    void setSiguiente(NodoLista *siguiente);
    NodoLista *getAnterior();
    void setAnterior(NodoLista *anterior);
    Transaccion *getTransaccion();
    void setTransaccion(Transaccion *transaccion);

};
```

- **NodoMatriz.h:** Es la definición de la Clase NodoMatriz en donde no tiene nada de especial con respecto a los métodos, pero que es la representación de un nodo dentro de la estructura de Matriz Dispersa.

```
class NodoMatriz {

private:
    NodoMatriz *siguiente;
    NodoMatriz *previo;

    NodoMatriz *arriba;
    NodoMatriz *abajo;

    NodoMatriz *adelante;
    NodoMatriz *atras;

    std::string nombreCabecera;
    Usuario *usuario;

    int id;
    int grupo;

public:
    static int contadorID;
    static int contadorGrupo;

    NodoMatriz(std::string nombreCabecera);
    NodoMatriz(Usuario *usuario);

    NodoMatriz *getSiguiente();
    void setSiguiente(NodoMatriz *siguiente);

    NodoMatriz *getPrevio();
    void setPrevio(NodoMatriz *previo);

    NodoMatriz *getArriba();
    void setArriba(NodoMatriz *arriba);

    NodoMatriz *getAbajo();
    void setAbajo(NodoMatriz *abajo);

    NodoMatriz *getAdelante();
    void setAdelante(NodoMatriz *adelante);
```

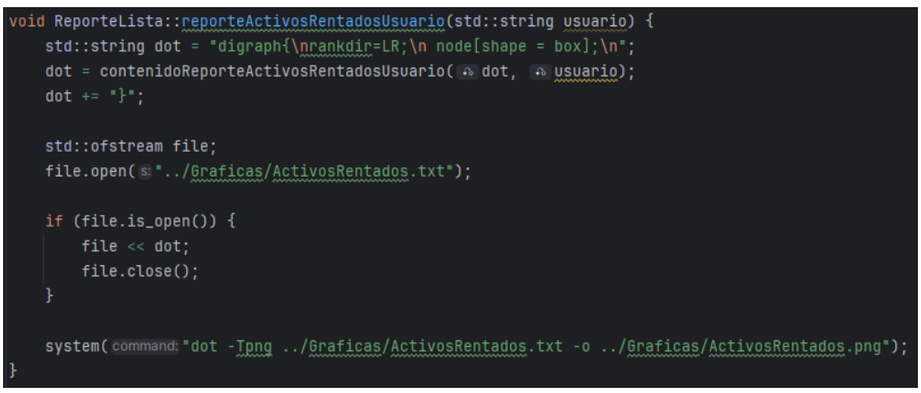
- **MatrizDispersa.cpp:** Clase que se encarga de Modelar la Estructura de Datos de una Matriz Dispersa, teniendo las operaciones básicas de una Matriz Dispersa así como métodos adicionales que ayudan al manejo de datos entre los usuario y los activos que se encuentran registrados en el programa.
- **ReporteMatriz.cpp:** Clase que se encarga de generar los reportes que se relacionan con la propia Matriz Dispersa, los reportes se generan con Graphviz por lo tanto habrá mucho código de Graphviz en esta clase.
    - reporteActivosEmpresa(string): Este método genera el reporte pertinente a los activos que se encuentran pertenecientes a una determinada empresa. El método se encarga de buscar la empresa en la que se desea consultar los activos, cuando la encuentra, entonces empieza recorrer todos los usuarios que estén dentro de la empresa, y en cada usuario se recorre su Árbol AVL para recoger los activos que tenga el usuario, y así irlos generando con código de Graphviz.

```
void ReporteMatriz::reporteActivosEmpresa(std::string empresa) {
    std::string dot = "digraph{\nrankdir = TB;\nlabel = \"Activos de una Empresa\";\nlabelloc = t;\nnode[style = filled];\nnode[shape=box color=red fillcolor=\"#8080F0\"];\n" + empresa + ";\n";
    NodoMatriz *nodoEmpresa = this->matrizDispersa->existeEmpresa( a: empresa);
    NodoMatriz *aux = nodoEmpresa->getSiguiente();
    NodoMatriz *aux2 = nullptr;
    while (aux != nullptr) {
        aux2 = aux;
        while (aux2 != nullptr) {
            dot += "node[shape=box color=transparent fillcolor=\"#808080\"];\n" + aux2->getUsuario()->getUsername() + ";\nnode[shape = circle];\n";
            dot += aux2->getUsuario()->getReporteArbol()->reporteActivosCabeceras( nodo: aux2->getUsuario()->getArbol()->getRaiz(), a: dot);
            aux2 = aux2->getAtras();
        }
        aux = aux->getSiguiente();
    }
    dot += "\n}";
    std::ofstream file;
    file.open( s: "../Graficas/ReporteActivosEmpresa.txt");
    if (file.is_open()) {
        file << dot;
        file.close();
    }
    system( command: "dot -Tpng ../Graficas/ReporteActivosEmpresa.txt -o ../Graficas/ReporteActivosEmpresa.png");
}
```

- reporteActivosDepartamento(string): Este método genera el reporte pertinente a los activos que se encuentran pertenecientes a un determinado departamento. El método se encarga de buscar el departamento en el que se desea consultar los activos, cuando lo encuentra, entonces empieza recorrer todos los usuarios que estén dentro del departamento, y en cada usuario se recorre su Árbol AVL para recoger los activos que tenga el usuario, y así irlos generando con código de Graphviz.

```
void ReporteMatriz::reporteActivosDepartamento(std::string departamento) {
    std::string dot = "digraph{\nrankdir = TB;\nlabel = \"Activos de un Departamento\";\nlabelloc = t;\nnode[style = filled];\nnode[shape=box color=red fillcolor=\"#8080F0\"];\n" + departamento + ";\n";
    NodoMatriz *nodoDepartamento = this->matrizDispersa->existeDepartamento( departamento);
    NodoMatriz *aux = nodoDepartamento->getAbajo();
    NodoMatriz *aux2 = nullptr;
    while (aux != nullptr) {
        aux2 = aux;
        while (aux2 != nullptr) {
            dot += "node[shape=box color=transparent fillcolor=\"#808080\"];\n" + aux2->getUsuario()->getUsername() + ";\n node[shape = circle];\n";
            dot += aux2->getUsuario()->getReporteArbol()->reporteActivosCabeceras( nodo: aux2->getUsuario()->getArbol()->getRaiz(),  dot);
            aux2 = aux2->getAtras();
        }
        aux = aux->getAbajo();
    }
    dot += "\n}";
    std::ofstream file;
    file.open( s: "../Graficas/ReporteActivosDepartamento.txt");
    if (file.is_open()) {
        file << dot;
        file.close();
    }
    system( command: "dot -Tpng ../Graficas/ReporteActivosDepartamento.txt -o ../Graficas/ReporteActivosDepartamento.png");
}
```

- **ListaTransacciones.cpp:** Clase que se encarga de Modelar la Estructura de Datos de una Lista Circular Doblemente Enlazada, teniendo las operaciones básicas de una Lista Circular Doblemente Enlazada, así como métodos adicionales que ayudan al manejo de datos entre los usuario y las transacciones que se encuentran registrados en el programa.
- **ReporteLista.cpp:** Clase que se encarga de generar los reportes que se relacionan con la propia Lista Circular Doblemente Enlazada, los reportes se generan con Graphviz por lo tanto habrá mucho código de Graphviz en esta clase.
    - reporteActivosRentadosUsuario(string): Metodo que se encarga de generar el reporte de los activos que ha rentado un determinado usuario, el contenido de código Graphviz lo recibe del método contenidoReporteActivosRentadosUsuario(string, string) el cual se explica a continuación

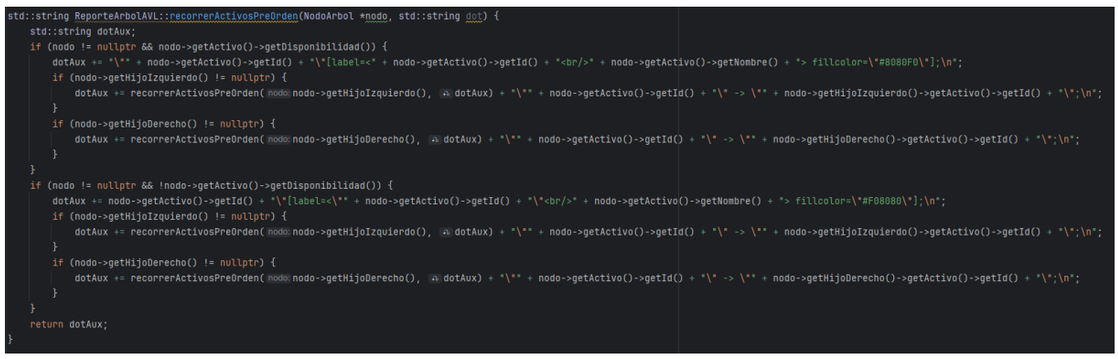


```
void ReporteLista::reporteActivosRentadosUsuario(std::string usuario) {
    std::string dot = "digraph{\nrankdir=LR;\n node[shape = box];\n";
    dot = contenidoReporteActivosRentadosUsuario( dot,  usuario);
    dot += "}";

    std::ofstream file;
    file.open( s: "../Graficas/ActivosRentados.txt");

    if (file.is_open()) {
        file << dot;
        file.close();
    }

    system( command: "dot -Tpng ../Graficas/ActivosRentados.txt -o ../Graficas/ActivosRentados.png");
}
```

    - contenidoReporteActivosRentadosUsuario(string, string): Este método se encarga de generar el código Graphviz necesario para poder representar el reporte de los activos rentados por un determinado usuario. En donde el método recorre la lista de transacciones y va buscando los nodos que corresponden a transacciones del usuario a reportar.

```
std::string ReporteLista::contenidoReporteActivosRentadosUsuario(std::string dot, std::string usuario) {
    int contador = 0;
    if (!this->listaTransacciones->isVacia()) {
        NodoLista *aux = this->listaTransacciones->getInicio();
        do {
            if (Utils::isEquals(aux->getTransaccion()->getUsuario(),  usuario) && (aux->getTransaccion()->getDias() != 0)) {
                contador++;
                dot += "activo" + std::to_string(contador) + "[label = <" + aux->getTransaccion()->getActivo()->getId() + "<br/>" + aux->getTransaccion()->getActivo()->getNombre() + ">]\n";
            }
            aux = aux->getSiguiente();
        } while (aux != this->listaTransacciones->getInicio());
        int contador2 = 0;
        do {
            if (Utils::isEquals(aux->getTransaccion()->getUsuario(),  usuario)
                && aux->getTransaccion()->getDias() != 0
                && contador2 != contador-1) {
                contador2++;
                dot += "activo" + std::to_string(contador2) + "->activo" + std::to_string( val: contador2+1) + "\n";
            }
            aux = aux->getSiguiente();
        } while (aux != this->listaTransacciones->getInicio());
        dot += usuario + "\n";
    } else {
        dot += "vacia;\n" + usuario + "\n";
    }
    return dot;
}
```

- **ArbolAVL.cpp:** Clase que se encarga de Modelar la Estructura de Datos de un Árbol AVL, teniendo las operaciones básicas del Árbol AVL así como métodos adicionales que ayudan al manejo de datos entre los usuario y los activos que se encuentran registrados en el programa.
- **ReporteArbolAVL.cpp:** Clase que se encarga de generar los reportes que se relacionan con el propio Árbol AVL, los reportes se generan con Graphviz por lo tanto habrá mucho código de Graphviz en esta clase.
    - recorrerActivosPreOrden(*nodo, string): Método que genera el código de Graphviz necesario para poder enlazar los nodos del Árbol AVL, dando el estilo a cada nodo dependiendo de la disponibilidad del activo en ese momento de recorrer el árbol.



```
std::string ReporteArbolAVL::recorrerActivosPreOrden(NodoArbol *nodo, std::string dot) {
    std::string dotAux;
    if (nodo != nullptr && nodo->getActivo()->getDisponibilidad()) {
        dotAux += "\"" + nodo->getActivo()->getId() + "\"[label=<" + nodo->getActivo()->getId() + "<br/>" + nodo->getActivo()->getNombre() + "> fillcolor=\"#8080F0\"];\n";
        if (nodo->getHijoIzquierdo() != nullptr) {
            dotAux += recorrerActivosPreOrden( nodo: nodo->getHijoIzquierdo(), dotAux) + "\"" + nodo->getActivo()->getId() + "\" -> \"" + nodo->getHijoIzquierdo()->getActivo()->getId() + "\";\n";
        }
        if (nodo->getHijoDerecho() != nullptr) {
            dotAux += recorrerActivosPreOrden( nodo: nodo->getHijoDerecho(), dotAux) + "\"" + nodo->getActivo()->getId() + "\" -> \"" + nodo->getHijoDerecho()->getActivo()->getId() + "\";\n";
        }
    }
    if (nodo != nullptr && !nodo->getActivo()->getDisponibilidad()) {
        dotAux += nodo->getActivo()->getId() + "\"[label=<\"" + nodo->getActivo()->getId() + "\"<br/>" + nodo->getActivo()->getNombre() + "> fillcolor=\"#F08080\"];\n";
        if (nodo->getHijoIzquierdo() != nullptr) {
            dotAux += recorrerActivosPreOrden( nodo: nodo->getHijoIzquierdo(), dotAux) + "\"" + nodo->getActivo()->getId() + "\" -> \"" + nodo->getHijoIzquierdo()->getActivo()->getId() + "\";\n";
        }
        if (nodo->getHijoDerecho() != nullptr) {
            dotAux += recorrerActivosPreOrden( nodo: nodo->getHijoDerecho(), dotAux) + "\"" + nodo->getActivo()->getId() + "\" -> \"" + nodo->getHijoDerecho()->getActivo()->getId() + "\";\n";
        }
    }
    return dotAux;
}
```

    - reporteActivosUsuario(*nodo, string): Metodo que genera el reporte de los activos de un determinado usuario, este método recibe el contenido principal del reporte gracias al método anteriormente mencionado de recorrerActivosPreOrden(*nodo, string), en donde este método ya solo recibe el contenido para generar el archivo de reporte como tal.

```
void ReporteArbolAVL::reporteActivosUsuario(NodoArbol *nodo, std::string username) {
    std::string dot = "digraph{\nrankdir = TB;\nlabel = \"Activos de un Usuario\";\nlabelloc = t;\nnode[color=transparent style=filled];\nnode[shape = box]\n" + username + "\nnode[shape = circle]; \n";
    dot += recorrerActivosPreOrden(nodo, dot) + "\n}";
    std::ofstream file;
    file.open( s: "../Graficas/ReporteActivosUsuario.txt");
    if (file.is_open()) {
        file << dot;
        file.close();
    }
    system( command: "dot -Tpng ../Graficas/ReporteActivosUsuario.txt -o ../Graficas/ReporteActivosUsuario.png");
    std::cout << ">> Reporte Generado!!!" << std::endl;
}
```

    - reporteActivosCabeceras(*nodo, string): Método que genera el contenido principal de código Graphviz para el reporte de activos de un departamento/empresa, este método le da cierto estilo a los nodos dependiendo de la disponibilidad del activo en ese momento, el método recorre los distintos nodos del Árbol AVL de manera recursiva.

```
std::string ReporteArbolAVL::reporteActivosCabeceras(NodoArbol *nodo, std::string dot) {
    std::string dotAux;
    if (nodo != nullptr && nodo->getActivo()->getDisponibilidad()) {
        dotAux += "\"" + nodo->getActivo()->getId() + "\"[label=<" + nodo->getActivo()->getId() + "<br/>" + nodo->getActivo()->getNombre() + "> fillcolor=\"#8080F0\"];\n";
        if (nodo->getHijoIzquierdo() != nullptr) {
            dotAux += reporteActivosCabeceras( nodo: nodo->getHijoIzquierdo(), dotAux) + "\"" + nodo->getActivo()->getId() + "\" -> \"" + nodo->getHijoIzquierdo()->getActivo()->getId() + "\";\n";
        }
        if (nodo->getHijoDerecho() != nullptr) {
            dotAux += reporteActivosCabeceras( nodo: nodo->getHijoDerecho(), dotAux) + "\"" + nodo->getActivo()->getId() + "\" -> \"" + nodo->getHijoDerecho()->getActivo()->getId() + "\";\n";
        }
    }
    if (nodo != nullptr && !nodo->getActivo()->getDisponibilidad()) {
        dotAux += nodo->getActivo()->getId() + "\"[label=<\"" + nodo->getActivo()->getId() + "\"<br/>" + nodo->getActivo()->getNombre() + "> fillcolor=\"#F08080\"];\n";
        if (nodo->getHijoIzquierdo() != nullptr) {
            dotAux += recorrerActivosPreOrden( nodo: nodo->getHijoIzquierdo(), dotAux) + "\"" + nodo->getActivo()->getId() + "\" -> \"" + nodo->getHijoIzquierdo()->getActivo()->getId() + "\";\n";
        }
        if (nodo->getHijoDerecho() != nullptr) {
            dotAux += recorrerActivosPreOrden( nodo: nodo->getHijoDerecho(), dotAux) + "\"" + nodo->getActivo()->getId() + "\" -> \"" + nodo->getHijoDerecho()->getActivo()->getId() + "\";\n";
        }
    }
    return dotAux;
}
```

DESCRIPCION DE TDAs

Lista Circular Doblemente Enlazada

- Descripción

Es una estructura donde el último elemento tiene como referencia siguiente al primer elemento, y la referencia al anterior del primer elemento de la lista también es el último. Es una especia de lista doblemente enlazada pero que posee una característica adicional para el desplazamiento dentro de la lista “esta no tiene fin” y tiene 2 apuntadores a si misma.

A través del uso de la lista doble podemos acceder a los datos recorriéndolos hacia adelante hasta el final o hacia atrás hasta el inicio. En una lista enlazada doblemente circular, cada nodo tiene dos enlaces, similares a los de la lista doblemente enlazada, excepto que el enlace anterior del primer nodo apunta al último y el enlace siguiente del último nodo, apunta al primero. Como en una lista doblemente enlazada, las inserciones y eliminaciones pueden ser hechas desde cualquier punto con acceso a algún nodo cercano.

Aunque estructuralmente una lista circular doblemente enlazada no tiene ni principio ni fin, un puntero de acceso externo puede establecer el nodo apuntador que está en la cabeza y así mantener el orden tan bien como en una lista doblemente enlazada. En las listas circulares dobles, nunca se llega a una posición en la que ya no sea posible desplazarse. Cuando se llega al último elemento, el desplazamiento volverá a comenzar desde el primer elemento.

- Operaciones Básicas
    - Insertar: El primer paso es crear un nodo para el dato que vamos a insertar. Si la lista esta vacía, o el valor del primer elemento de la lista es mayor que el del nuevo, insertamos el nuevo nodo en la primera posición de la lista. En caso contrario, buscamos el lugar adecuado para la inserción, tenemos un puntero “anterior”. Lo inicializamos con el valor de la lista, y avanzaremos mientras anterior -> siguiente no sea nullptr y el dato que contiene anterior -> siguiente sea menor o igual que el dato que queremos insertar. Ahora ya tenemos anterior señalando al nodo adecuado, así que insertamos el nuevo nodo a continuación de él.
    - Buscar: A la hora de buscar elementos en una lista circular solo hay que tener una precaución, es necesario almacenar el puntero del nodo en que se empezó a la búsqueda, para poder detectar el caso en que no exista el valor que se busca. Por lo demás, a búsqueda es igual que en el caso de las listas abiertas, salvo que podemos empezar en cualquier punto de la lista.

Matriz Dispersa

- Descripción

Es una matriz de gran tamaño donde la mayoría de sus elementos son ceros o nulos, con esto se puede sacar ventaja a la hora de realizar una inserción o búsqueda ya que solo estaremos creando los elementos que necesitamos. La matriz dispersa consta de dos partes, la primer parte son los nodos encabezados que contienen nuestros ejes vertical y horizontal, y la segunda parte de los nodos ortogonales que contienen el valor que almacenaremos en la matriz y la coordenada donde se almacena.

- Operaciones Básicas
    - Insertar: Antes que nada se verifican si los encabezados existen, de ser así entonces se crean los encabezados correspondientes, en caso que uno de los encabezados exista, entonces solo se crea el encabezado faltante. Teniendo los encabezados creados, se procede a crear el nodo ortogonal, para después en nuestros encabezados crear los apuntadores a este nuevo nodo ortogonal y de igual forma se crean los apuntadores del nodo ortogonal hacia los dos encabezados que le corresponden.
    - Búsqueda:

Árbol AVL

- Descripción

Un árbol AVL es un tipo especial de árbol binario, fue el primer árbol de búsqueda auto-balanceado que se ideo. Los arboles 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 o viceversa. Gracias a esta forma de equilibrio, 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.

Para conseguir esta propiedad de equilibrio, la inserción y eliminación de los nodos se ha de realizar de una forma especial. Si al realizar una operación de inserción o eliminación se rompe la condición de equilibrio, hay que realizar una serie de rotaciones de los nodos.

- Operaciones Básicas
    - Insertar: La inserción en un árbol AVL puede ser realizada insertando el valor dado en el árbol como si fuera un árbol de búsqueda binario desequilibrado y después retrocediendo hacia la raíz, rotando sobre cualquier nodo que pueda haberse desequilibrado durante la inserción. Para la inserción se busca hasta encontrar la posición de la inserción o modificación, luego se inserta el nuevo nodo con factor de equilibrio, verificando el equilibrio de los nodos y re-equilibrando si es necesario. Debido a que las rotaciones son una operación que tienen complejidad constante y a que la altura está limitada a O(log n), el tiempo de ejecución para la inserción es del orden O(log n).
    - Eliminar: El procedimiento de eliminación es el mismo que en un árbol binario de búsqueda. La diferencia se encuentra en el proceso de reequilibrio posterior. Una extracción trae consigo una disminución de la altura de la rama donde se extrajo y tendrá como efecto un cambio en el factor de equilibrio del nodo padre de la rama en cuestión, pudiendo necesitarse una rotación simple o rotación doble.
    - Rotaciones: El reequilibrio se produce de abajo hacia arriba sobre los nodos en los que se produce el desequilibrio. Pueden darse dos casos: rotación simple o rotación doble; a su vez ambos casos pueden ser hacia la derecha o hacia la izquierda.
        - Rotación Simple a la Derecha: Esta rotación se usara cuando el subárbol izquierdo de un nodo sea 2 unidades más alto que el derecho, es decir, cuando su Factor de Equilibrio sea de -2. Y además, la raíz del subárbol izquierdo tenga un Factor de Equilibrio de -1 o 0, es decir, que este cargado a la izquierda o equilibrado.
        - Rotación Simple a la Izquierda: Se trata de caso simétrico de anterior. Esta rotación se usara cuando el subárbol derecho de un nodo sea 2 unidades más alto que el izquierdo, es decir, cuando su Factor de equilibrio sea de 2. Y además, la raíz del subárbol derecho tenga un Factor de Equilibrio de 1 o 0, es decir, que este cargado a la derecha o este equilibrado.
        - Rotación Doble a la Derecha: Esta rotación se usara cuando el subárbol izquierdo de un nodo sea 2 unidades más alto que el derecho, es decir, cuando su Factor de Equilibrio sea de -2. Y además, la raíz del subárbol izquierdo tenga Factor de Equilibrio de 1, es decir, que este cargado a la derecha.
        - Rotación Doble a la Izquierda: Esa rotación se usara cuando el subárbol derecho de un nodo sea 2 unidades más alto que el izquierdo, es decir, cuando su Factor de Equilibrio sea de 2: Y además, la raíz del subárbol derecho tenga un Factor de Equilibrio de -1, es decir, que este cargado a la izquierda: Se trata del caso simétrico del anterior.
    - Búsqueda: El procedimiento es casi similar a un Árbol de Búsqueda Binaria. Sin embargo por ser un árbol ordenado y equilibrado no se tomara mucho tiempo para hallar el elemento deseado.