UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

MANUAL TECNICO

ESTRUCTURA DE DATOS

FACULTAD DE INGENIERIA

GUATEMELA, FEBRERO 2024

INDICE

INTRODUCCIÓN

Se utilizó el lenguaje de programación Fortran para implementar y gestionar eficientemente las estructuras de datos lineales, aprovechando las características específicas del lenguaje en el desarrollo de la aplicación, la cual consiste en una imprenta de pixel arts. La aplicación representa visualmente las estructuras mediante biblioteca Graphviz.

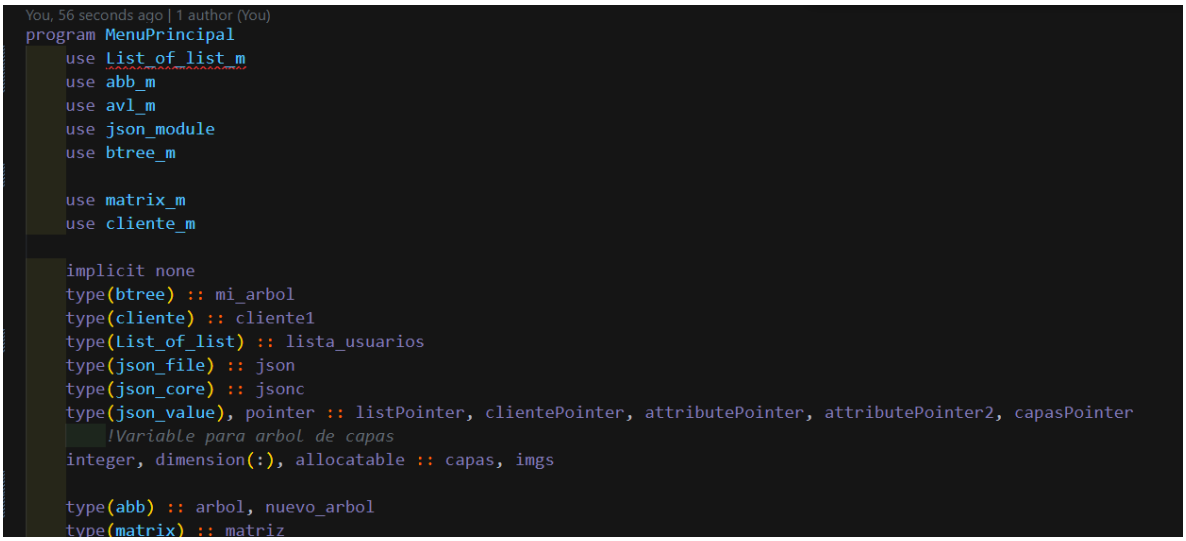
ESPECIFICACIONES DEL SISTEMA

El programa fue desarrollado en el lenguaje Fortran y su manejador de programas fpm para facilitar el orden dentro de la compilación. Además, integrar la herramienta Graphviz para generar representaciones visuales claras y comprensibles de las estructuras de datos lineales utilizadas en la simulación, facilitando así la comprensión del funcionamiento interno de la aplicación.

Guía del programa

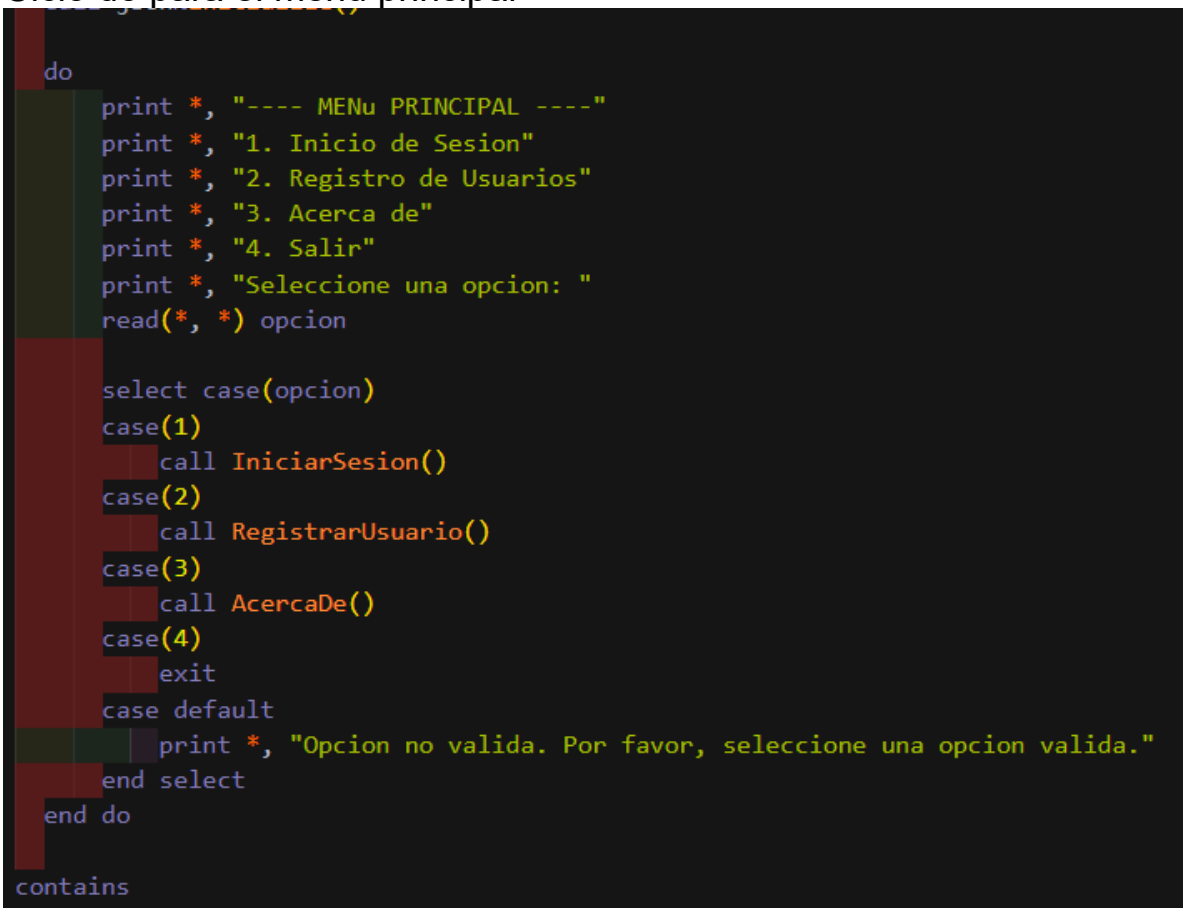
Main.f90:

Se declaran todas las variables a utilizar:



```
You, 56 seconds ago | 1 author (You)
program MenuPrincipal
    use List_of_list_m
    use abb_m
    use avl_m
    use json_module
    use btree_m

    use matrix_m
    use cliente_m

    implicit none
    type(btree) :: mi_arbol
    type(cliente) :: cliente1
    type(List_of_list) :: lista_usuarios
    type(json_file) :: json
    type(json_core) :: jsonc
    type(json_value), pointer :: listPointer, clientePointer, attributePointer, attributePointer2, capasPointer
        !Variable para arbol de capas
    integer, dimension(:), allocatable :: capas, imgs

    type(abb) :: arbol, nuevo_arbol
    type(matrix) :: matriz
```

Ciclo do para el menú principal



```
do
    print *, "---- MENu PRINCIPAL ----"
    print *, "1. Inicio de Sesion"
    print *, "2. Registro de Usuarios"
    print *, "3. Acerca de"
    print *, "4. Salir"
    print *, "Seleccione una opcion: "
    read(*, *) opcion

    select case(opcion)
    case(1)
        call IniciarSesion()
    case(2)
        call RegistrarUsuario()
    case(3)
        call AcercaDe()
    case(4)
        exit
    case default
        print *, "Opcion no valida. Por favor, seleccione una opcion valida."
    end select
end do

contains
```

Función para pasar de str a int

```
function string_to_integer(str)
    character(len=*), intent(in) :: str
    integer :: string_to_integer
    integer(kind=8) :: i, num       You, 2 minutes ago • Uncommitted changes

    ! Inicializar num a cero
    num = 0

    do i = 1, len(str)
        if (str(i:i) >= '0' .and. str(i:i) <= '9') then
            num = num * 10 + ichar(str(i:i)) - ichar('0')
        else
            ! Si se encuentra un carácter no numérico, imprimir el mensaje de error y salir de la funcion
            print *, "Error: El string contiene caracteres no numéricos."
            string_to_integer = -1
            return
        end if
    end do

    ! Devolver el número solo si la conversion se realizo correctamente
    string_to_integer = num
end function string_to_integer
```

Carga Masiva: Se lee el json seleccionado por el usuario y se carga al árbol o lista según sea el caso

```
subroutine carga_masiva_usuarios()
    integer(kind=8):: int_dpi
    print *, 'Ha seleccionado Carga masiva de clientes'
    ! read the file
    print *, 'Ingrese el nombre del  archivo JSON:'
    read(*, '(A)') nombre_json

    call json%load(filename=nombre_json)

    ! print the file to the console
    call json%print()

    call json%info('', n_children=size)

    call json%get_core(jsonc)
    call json%get('', listPointer, found)

    do i = 1, size
      call jsonc%get_child(listPointer, i, clientePointer, found)

      !imprimiendo el atributo nombre_cliente
      call jsonc%get_child(clientePointer, 'nombre_cliente', attributePointer, found)
      call jsonc%get(attributePointer, nombre_cliente)
      print *, "Nombre del cliente: ", trim(nombre_cliente)
```

```
        call jsonc%get_child(clientePointer, 'nombre_cliente', attributePointer, found)
        call jsonc%get(attributePointer, nombre_cliente)
        print *, "Nombre del cliente: ", trim(nombre_cliente)

        !imprimiendo el atributo dpi
        call jsonc%get_child(clientePointer, 'dpi', attributePointer, found)
        call jsonc%get(attributePointer, dpi)
        print *, "DPI del cliente: ", dpi

        !imprimiendo el atributo password
        call jsonc%get_child(clientePointer, 'password', attributePointer, found)
        call jsonc%get(attributePointer, password)
          print *, "Contrasena del cliente: ", trim(password)
        int_dpi = string_to_integer(dpi)

          print *, "DPI del cliente (entero): ", int_dpi
        cliente1 = cliente(dpi=int_dpi, nombre=nombre_cliente, contrasena=password)


        call mi_arbol%insert(cliente1)

      end do
  end subroutine carga_masiva_usuarios
```

Inicio de sesión: Busca primero si es el administrador, y luego lo busca en el árbol b, si no se encuentra no deja iniciar sesión.

```
subroutine IniciarSesion()
    implicit none
    character(len=20) :: usuario, contrasena
    logical :: login_successful

    ! Logica para el inicio de sesion
    print *, "Ingrese el nombre de usuario: "
    read(*, *) usuario
    print *, "Ingrese la contrasena: "
    read(*, *) contrasena

    ! Comprobamos si el usuario es el administrador
    if (usuario == 'admin' .and. contrasena == 'EDD2024') then
        print *, "Inicio de sesion exitoso como administrador."
        call MenuAdministrador()
        return
    end if
    print *, "No es el administrador."
    ! Buscamos el usuario en la lista de usuarios
    login_successful = mi_arbol%buscar_usuario(usuario, contrasena)

    ! Comprobar si el inicio de sesion fue exitoso
    if (login_successful) then
        print *, "Inicio de sesion exitoso."
        call MenuUsuario(usuario)
        return
    else
        print *, "Nombre de usuario o contrasena incorrectos."
    end if
    ! Si no se encontro el usuario en la lista, se muestra un mensaje de error
    print *, "Nombre de usuario o contrasena incorrectos."
end subroutine IniciarSesion
```

Menu usuario: Visualiza el menú del usuario

```
subroutine MenuUsuario(usuario)
    implicit none
    character(len=20), intent(in) :: usuario

    integer :: opcion

    do
        call MostrarMenuUsuario(usuario)
        read(*, *) opcion

        select case(opcion)
        case(1)
            call reportes_estructuras()
        case(2)
            call gestion_imagenes()
        case(3)
            call opciones_carga_masiva()
        case(4)
            exit
        case default
            print *, "Opcion no valida. Por favor, seleccione una opcion valida."
        end select
    end do
    end subroutine MenuUsuario

    subroutine MostrarMenuUsuario(usuario)
        implicit none
        character(len=20), intent(in) :: usuario
        print *, "---- MENu Usuario ----" // usuario
        print *, "1. Visualizar reportes de las estructuras"
        print *, "2. Navegacion y gestion de imagenes."
        print *, "3. Opciones de carga masiva."
        print *, "4. Salir."
        print *, "Seleccione una opcion: "
    end subroutine MostrarMenuUsuario
```

Modificar y eliminar usuario

```
subroutine ModificarUsuario()
    implicit none
character(len=20) :: nombre, password
character(len=20) :: nombre_nuevo, password_nuevo
print *, "Modificar Usuario."
print *, "Ingrese el nombre completo del  usuario: "
read(*, '(A)') nombre
print *, "Ingrese la contraseña del usuario: "
read(*, '(A)') password
print *, "Ingrese el nuevo nombre del usuario: "
read(*, '(A)') nombre_nuevo
print *, "Ingrese la nueva contraseña del usuario: "
read(*, '(A)') password_nuevo
call mi_arbol%modificar_usuario(nombre, password, nombre_nuevo, password_nuevo)


end subroutine ModificarUsuario

subroutine EliminarUsuario()
    implicit none
character(len=20) :: nombre, password
print *, "Eliminar Usuario."
print *, "Ingrese el nombre completo del  usuario: "
read(*, '(A)') nombre
print *, "Ingrese la contraseña del usuario: "
read(*, '(A)') password
call mi_arbol%eliminar_usuario(nombre, password)
end subroutine EliminarUsuario
```

Menu admin

```
subroutine MenuAdministrador()
    implicit none
    integer :: opcion

    do
        call MostrarMenuAdministrador()
        read(*, *) opcion

        select case(opcion)
        case(1)
            call grafico_arbol_usuarios()
        case(2)
            call operaciones_usuarios()
        case(3)
            call carga_masiva_usuarios()
       case(4)
            call reportes_Admin()
       case(5)
            exit
        case default
            print *, "Opcion no valida. Por favor, seleccione una opcion valida."
        end select
    end do
end subroutine MenuAdministrador
```

Reportes admin

```
subroutine reporte_uno()
  implicit none
  integer(kind=8):: int_dpi

  print *, "Ingrese el DPI del usuario a reportar: "
  read*,int_dpi
  call mi_arbol%buscarCliente(int_dpi)
end subroutine reporte_uno

subroutine Mostrar_reportes_Admin()
  print *, "Reportes del Administrador."
    print *, "1. Reporte un usuario."
      print *, "2. Reporte todos los usuarios."
      print *, "3. Salir."
      print *, "Seleccione una opcion: "
  end subroutine Mostrar_reportes_Admin
subroutine RegistrarUsuario()
  implicit none
  character(len=40) :: nombre_completo, password_usuario
  integer(kind=8):: int_dpi2
    ! Logica para el registro de usuarios
    print *, "Ingrese el nombre completo del nuevo usuario: "
    read(*, '(A)') nombre_completo
    print *, "Ingrese el DPI del nuevo usuario: "
    read*,int_dpi2
    print *, "Ingrese la contrasena para el nuevo usuario: "
    read(*, '(A)') password_usuario

    cliente1 = cliente(dpi=int_dpi2, nombre=nombre_completo, contrasena=password_usuario)
        call mi_arbol%insert(cliente1)
end subroutine RegistrarUsuario
```

Reportes estructuras:

```
end subroutine AcercaDe

subroutine reportes_estructuras()
      implicit none
      integer :: opcion
      do
          call MostrarMenuReportes()
          read(*, *) opcion

          select case(opcion)
          case(1)
              call arbolavl%graficar()
          case(2)
              call arbol%graph_abb("capas")
          case(3)
              call listarCapas()
          case(4)
              exit
          case default
              print *, "Opcion no valida. Por favor, seleccione una opcion valida."
          end select
      end do
end subroutine reportes_estructuras
```

Menu opciones de carga masiva

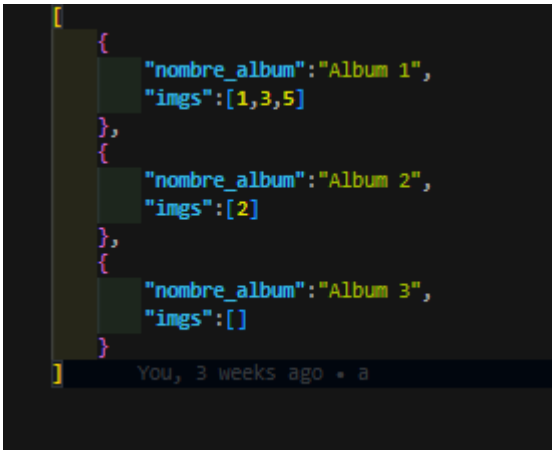
```
subroutine opciones_carga_masiva()
    implicit none
    integer :: opcion

    do
        call MostrarMenuCargaMasiva()
        read(*, *) opcion

        select case(opcion)
        case(1)
            call CargaCapas()
        case(2)
            call CargaImagenes()
        case(3)
            call CargaAlbumes()
        case(4)
            exit
        case default
            print *, "Opcion no valida. Por favor, seleccione una opcion valida."
        end select
    end do
end subroutine opciones_carga_masiva
```

Archivos json ejemplo

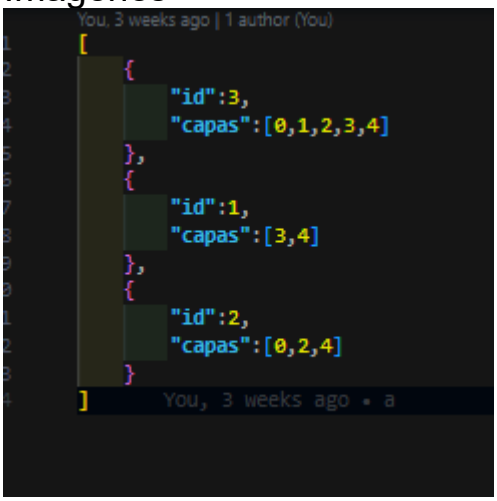
Clientes


```
[
    {
        "dpi":"1",
        "nombre_cliente":"AUX EDD",
        "password":"edd1s2022"
    },
    {
        "dpi":"2",
        "nombre_cliente":"cliente 2",
        "password":"edd1s2022"
    },
    {
        "dpi":"3",
        "nombre_cliente":"cliente 3",
        "password":"edd1s2022"
    },
    {
        "dpi":"4",
        "nombre_cliente":"cliente 4",
        "password":"edd1s2022"
    },
    {
        "dpi":"5",
        "nombre_cliente":"cliente 5",
        "password":"edd1s2022"
    },
    {
        "dpi":"6",
        "nombre_cliente":"cliente 6",
        "password":"edd1s2022"
    },
    {
        "dpi":"7",
        "nombre_cliente":"cliente 7",
        "password":"edd1s2022"
    },      You, 8 hours ago • Uncommitted changes
    {
        "dpi":"8",
```

Albumes



Imagenes



Capas

Módulos

Abb: modulo para el árbol abb tiene la clase pixel para guardar cada pixel en la imagen, el cual guarda el color, fila y columna.

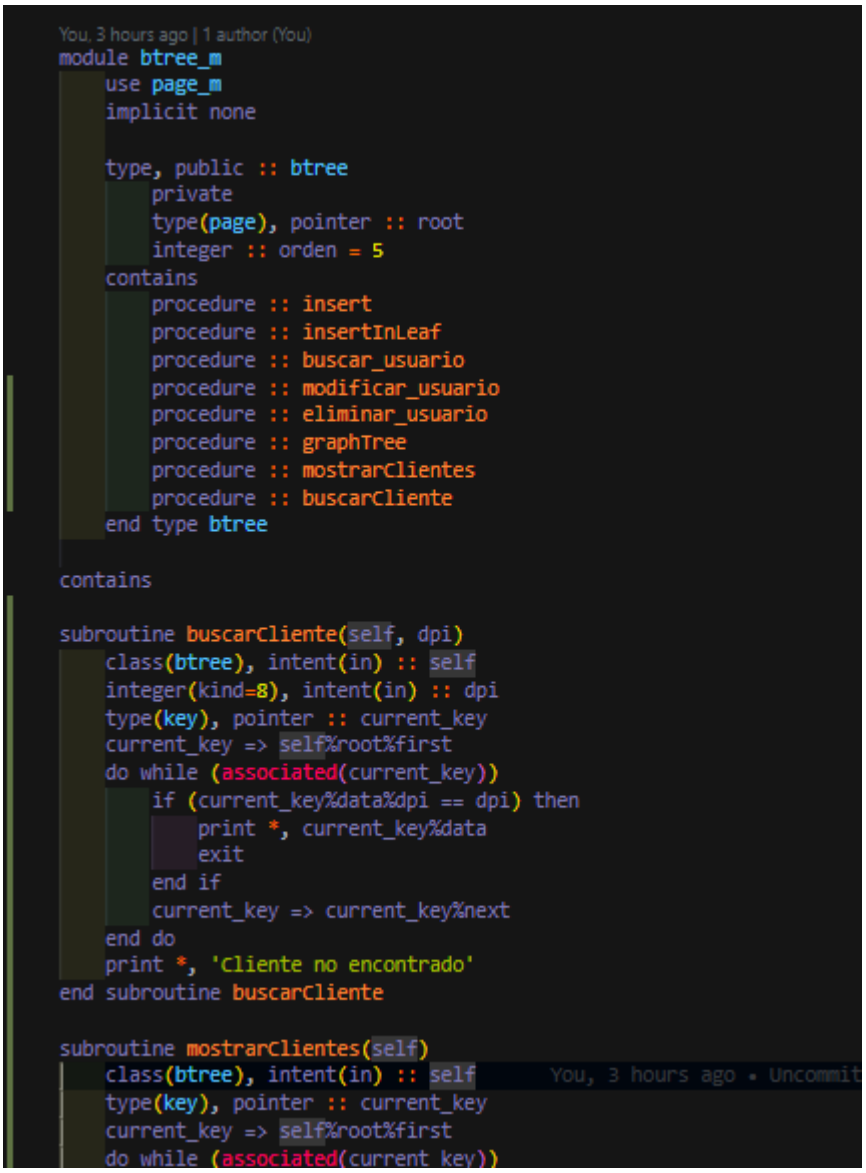
```
You, 1 hour ago | 1 author (You)
module abb_m
    use uuid_module
    implicit none

    type :: pixel
        integer :: fila
        integer :: columna
        character(len=7) :: color
    end type pixel

type :: node
    integer :: value
    integer :: height = 1
    type(node), pointer :: right => null()
    type(node), pointer :: left => null()
    type(pixel), allocatable :: pixeles(:)
end type node
    You, 3 weeks ago • Registro usuario e inicio de sesi
    type, public :: abb
        type(node), pointer :: root => null()
    contains
        procedure :: insert_abb
        procedure :: delete_abb
        procedure :: preorder_abb
        procedure :: buscar_abb
        procedure :: inorder_abb
        procedure :: posorder_abb
        procedure :: graph_abb
        procedure :: buscarCapa_abb
        procedure :: inOrder
        procedure :: preOrder
        procedure :: posOrder
        procedure :: amplitudOrden
        procedure :: buscar
    end type abb
contains

subroutine buscar(self, val)
    class(abb), intent(in) :: self
    integer, intent(in) :: val
    type(pixel) :: info
```

```
subroutine buscar(self, val)
    class(abb), intent(in) :: self
    integer, intent(in) :: val
    type(pixel) :: info

    ! Inicializamos la información con valores por defecto
    info%fila = -1
    info%columna = -1
    info%color = 'Ninguno'

    ! Realizamos la búsqueda en el árbol
    if (associated(self%root)) then
        call buscarRec_abb(self%root, val, info)
    end if
end subroutine buscar
```

```
recursive subroutine buscarRec_abb(root, val, info)
    type(node), pointer :: root
    integer, intent(in) :: val
    type(pixel), intent(inout) :: info

    if (associated(root)) then
        if (val == root%value) then
            ! Si encontramos el valor, actualizamos la información
            if (size(root%pixeles) > 0) then
                info = root%pixeles(1)
            end if
        else if (val < root%value) then
            ! Si el valor buscado es menor, buscamos en el subárbol izquierdo
            call buscarRec_abb(root%left, val, info)
        else
            ! Si el valor buscado es mayor, buscamos en el subárbol derecho
            call buscarRec_abb(root%right, val, info)
        end if
    end if
end subroutine buscarRec_abb


    !Subrutinas de apoyo
    recursive subroutine insertRec_abb(root, val, info)
        type(node), pointer, intent(inout) :: root
        integer, intent(in) :: val
        type(pixel), intent(in) :: info

        if (val < root%value) then
            if (.not. associated(root%left)) then
                allocate(root%left)
                root%left%value = val
                allocate(root%left%pixeles(1))
                root%left%pixeles(1) = info
            else
                call insertRec_abb(root%left, val, info)
            end if
        else if (val > root%value) then
            if (.not. associated(root%right)) then
                allocate(root%right)
                root%right%value = val
                allocate(root%right%pixeles(1))
                root%right%pixeles(1) = info
            else
                call insertRec_abb(root%right, val, info)
            end if
        end if
    end subroutine insertRec_abb
```

```
recursive function deleteRec_abb(root, key) result(res)
    type(node), pointer :: root
    integer, intent(in) :: key
    type(node), pointer :: res
    type(node), pointer :: temp

    if (.not. associated(root)) then
        res => root
        return
    end if

    if (key < root%value) then
        root%left => deleteRec_abb(root%left, key)
    else if (key > root%value) then
        root%right => deleteRec_abb(root%right, key)
    else
        if (.not. associated(root%left)) then
            temp => root%right
            deallocate(root)
            res => temp
            return
        else if (.not. associated(root%right)) then
            temp => root%left
            deallocate(root)
            res => temp
            return
        else
            call getMajorOfMinors_abb(root%left, temp)
            root%value = temp%value
            root%left => deleteRec_abb(root%left, temp%value)
        end if
    end if

    res => root
end function deleteRec_abb
```

Arbolb: este modulo guarda el árbol b de usuarios, no funciona si se ingresan 5 o mas usuarios. Usa el modulo de page para guardar la estructura

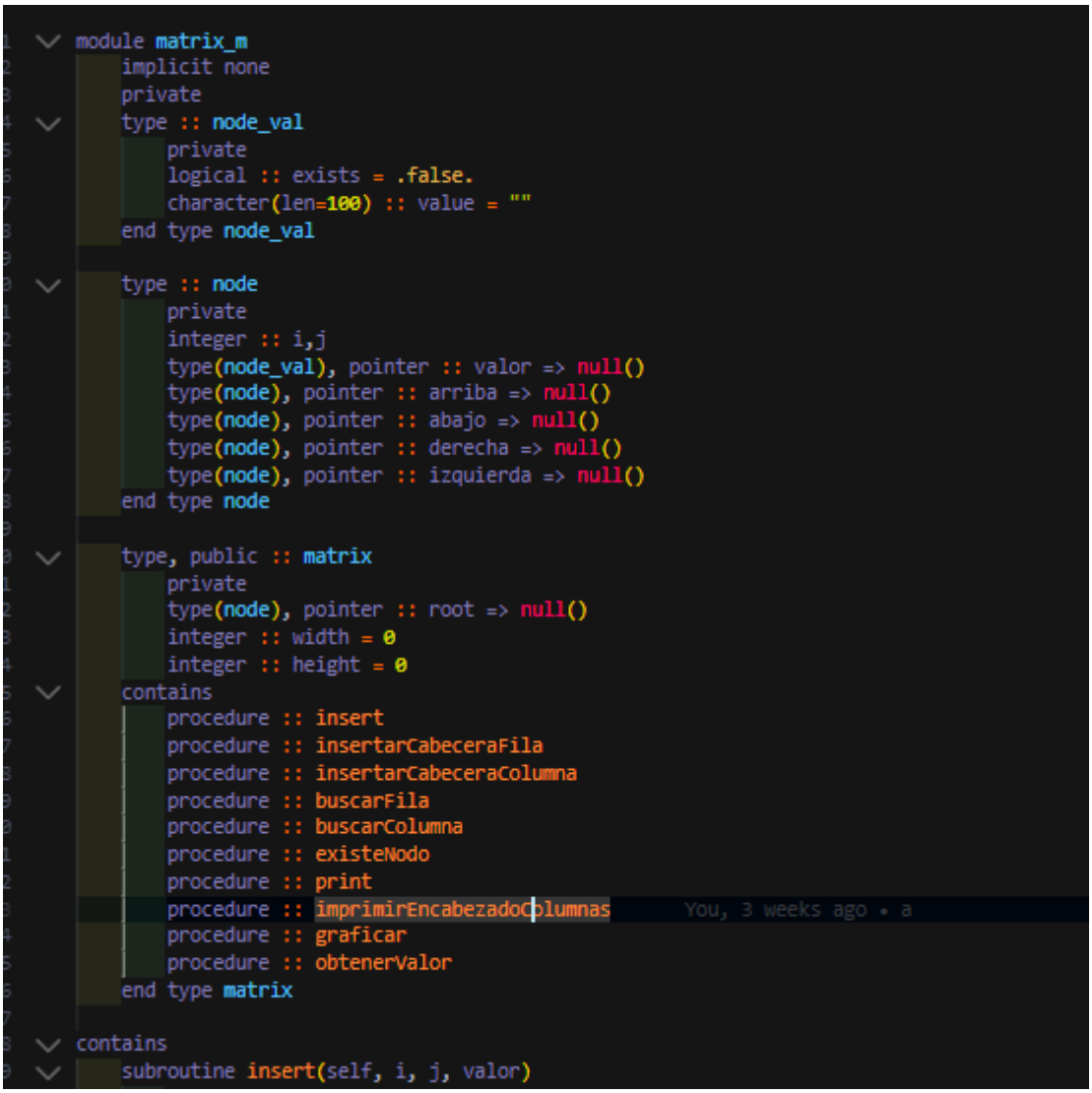


```
You, 3 hours ago | 1 author (You)
module btree_m
    use page_m
    implicit none

    type, public :: btree
        private
        type(page), pointer :: root
        integer :: orden = 5
    contains
        procedure :: insert
        procedure :: insertInLeaf
        procedure :: buscar_usuario
        procedure :: modificar_usuario
        procedure :: eliminar_usuario
        procedure :: graphTree
        procedure :: mostrarClientes
        procedure :: buscarCliente
    end type btree

contains

subroutine buscarCliente(self, dpi)
    class(btree), intent(in) :: self
    integer(kind=8), intent(in) :: dpi
    type(key), pointer :: current_key
    current_key => self%root%first
    do while (associated(current_key))
        if (current_key%data%dpi == dpi) then
            print *, current_key%data
            exit
        end if
        current_key => current_key%next
    end do
    print *, 'Cliente no encontrado'
end subroutine buscarCliente

subroutine mostrarClientes(self)
    class(btree), intent(in) :: self        You, 3 hours ago • Uncommit
    type(key), pointer :: current_key
    current_key => self%root%first
    do while (associated(current_key))
```

Page

```
You, 3 hours ago | 1 author (You)
module cliente_m
    implicit none

    ! Definición del tipo cliente
    type :: cliente
        integer(kind=16) :: dpi      You, 3 hours ago •
        character(len=50) :: nombre
        character(len=20) :: contrasena
    end type cliente
end module cliente_m

module page_m
    use cliente_m
    implicit none

    type, public :: page
        logical :: leaf = .true.
        type(key), pointer :: first => null()
        integer :: numberKeys = 0

    contains
        procedure :: insertKey
    end type page

    type :: key
        type(cliente) :: data
        type(key), pointer :: prev => null()
        type(key), pointer :: next => null()
        type(page), pointer :: right => null()
        type(page), pointer :: left => null()
    contains
        procedure :: hasKids
    end type key
```

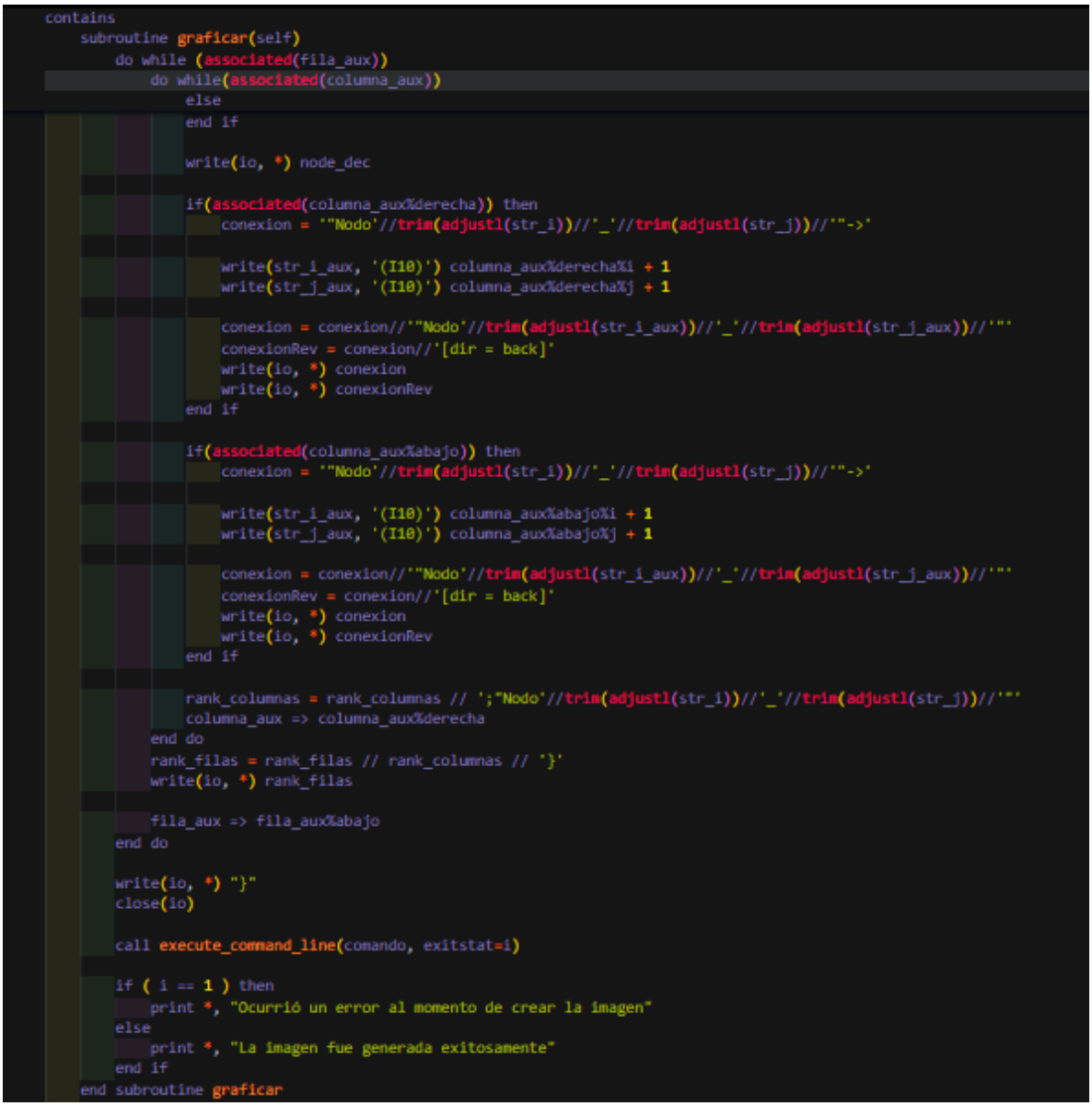
Matriz: este modulo guarda la matriz dispersa, y la grafica, este modulo no fue implementado.



```
module matrix_m
    implicit none
    private
    type :: node_val
        private
        logical :: exists = .false.
        character(len=100) :: value = ""
    end type node_val

    type :: node
        private
        integer :: i,j
        type(node_val), pointer :: valor => null()
        type(node), pointer :: arriba => null()
        type(node), pointer :: abajo => null()
        type(node), pointer :: derecha => null()
        type(node), pointer :: izquierda => null()
    end type node

    type, public :: matrix
        private
        type(node), pointer :: root => null()
        integer :: width = 0
        integer :: height = 0
    contains
        procedure :: insert
        procedure :: insertarCabeceraFila
        procedure :: insertarCabeceraColumna
        procedure :: buscarFila
        procedure :: buscarColumna
        procedure :: existeNodo
        procedure :: print
        procedure :: imprimirEncabezadoColumnas      You, 3 weeks ago • a
        procedure :: graficar
        procedure :: obtenerValor
    end type matrix

contains
    subroutine insert(self, i, j, valor)
```

```
contains
    subroutine insert(self, i, j, valor)
        class(matrix), intent(inout) :: self
        integer, intent(in) :: i
        integer, intent(in) :: j
        character(len=*), intent(in) :: valor

        type(node), pointer :: nuevo
        type(node), pointer :: fila
        type(node), pointer :: columna

        allocate(nuevo)
        allocate(nuevo%valor)
        nuevo%i = i
        nuevo%j = j
        nuevo%valor%value = valor

        if(.not. associated(self%root)) then
            allocate(self%root)
            self%root = node(i=-1, j=-1)
        end if

        fila => self%buscarFila(j)
        columna => self%buscarColumna(i)

        if(i > self%width) self%width = i
        if(j > self%height) self%height = j

        if(.not. self%existeNodo(nuevo)) then
            if(.not. associated(columna)) then
                columna => self%insertarCabeceraColumna(i)
            end if

            if(.not. associated(fila)) then
                fila => self%insertarCabeceraFila(j)
            end if
            call insertarEnColumna(nuevo, fila)
            call insertarEnFila(nuevo, columna)
        end if
    end subroutine insert
```

```
subroutine graficar(self)
    class(matrix), intent(in) :: self

    integer :: io
    integer :: i
    character(len=10) :: str_i
    character(len=10) :: str_j
    character(len=10) :: str_i_aux
    character(len=10) :: str_j_aux
    character(len=150) :: node_dec
    character(len=20) :: nombre

    character(len=100) :: comando
    character(len=50) :: contenido
    character(:), allocatable :: rank_filas
    character(:), allocatable :: rank_columnas
    character(:), allocatable :: conexion
    character(:), allocatable :: conexionRev
    type(node), pointer :: fila_aux
    type(node), pointer :: columna_aux
    io = 1
    fila_aux => self%root
    comando = "dot -Tpng ./matrix.dot -o ./matrix.png"

    open(newunit=io, file="./matrix.dot")

    write(io, *) "digraph Matrix {"
    write(io, *) 'node[shape = "box", width=0.5, height=0.5]'
    write(io, *) 'rankdir=LR'

    do while (associated(fila_aux))
        rank_filas = "{rank=same"
        columna_aux => fila_aux
        rank_columnas = ""
        do while(associated(columna_aux))
            write(str_i, '(I10)') columna_aux%i + 1
            write(str_j, '(I10)') columna_aux%j + 1
            nombre = '"Nodo'//trim(adjustl(str_i))//'_'//trim(adjustl(str_j))//'"'

            if (columna_aux%i == -1 .and. columna_aux%j == -1) then
                node_dec = trim(adjustl(nombre))//'[label = "root", group="'//trim(adjustl(str_i))//'"]'
            else if(columna_aux%i == -1) then
```



```
contains
    subroutine graficar(self)
        do while (associated(fila_aux))
            do while(associated(columna_aux))
                else
                end if

                write(io, *) node_dec

                if(associated(columna_aux%derecha)) then
                    conexion = '"Nodo'//trim(adjustl(str_i))//'_'//trim(adjustl(str_j))//'"->'

                    write(str_i_aux, '(I10)') columna_aux%derecha%i + 1
                    write(str_j_aux, '(I10)') columna_aux%derecha%j + 1

                    conexion = conexion//'"Nodo'//trim(adjustl(str_i_aux))//'_'//trim(adjustl(str_j_aux))//'"'
                    conexionRev = conexion//'[dir = back]'
                    write(io, *) conexion
                    write(io, *) conexionRev
                end if

                if(associated(columna_aux%abajo)) then
                    conexion = '"Nodo'//trim(adjustl(str_i))//'_'//trim(adjustl(str_j))//'"->'

                    write(str_i_aux, '(I10)') columna_aux%abajo%i + 1
                    write(str_j_aux, '(I10)') columna_aux%abajo%j + 1

                    conexion = conexion//'"Nodo'//trim(adjustl(str_i_aux))//'_'//trim(adjustl(str_j_aux))//'"'
                    conexionRev = conexion//'[dir = back]'
                    write(io, *) conexion
                    write(io, *) conexionRev
                end if

                rank_columnas = rank_columnas // ';"Nodo'//trim(adjustl(str_i))//'_'//trim(adjustl(str_j))//'"'
                columna_aux => columna_aux%derecha
            end do
            rank_filas = rank_filas // rank_columnas // '}'
            write(io, *) rank_filas

            fila_aux => fila_aux%abajo
        end do

        write(io, *) "}"
        close(io)

        call execute_command_line(comando, exitstat=i)

        if ( i == 1 ) then
            print *, "Ocurrió un error al momento de crear la imagen"
        else
            print *, "La imagen fue generada exitosamente"
        end if
    end subroutine graficar
```

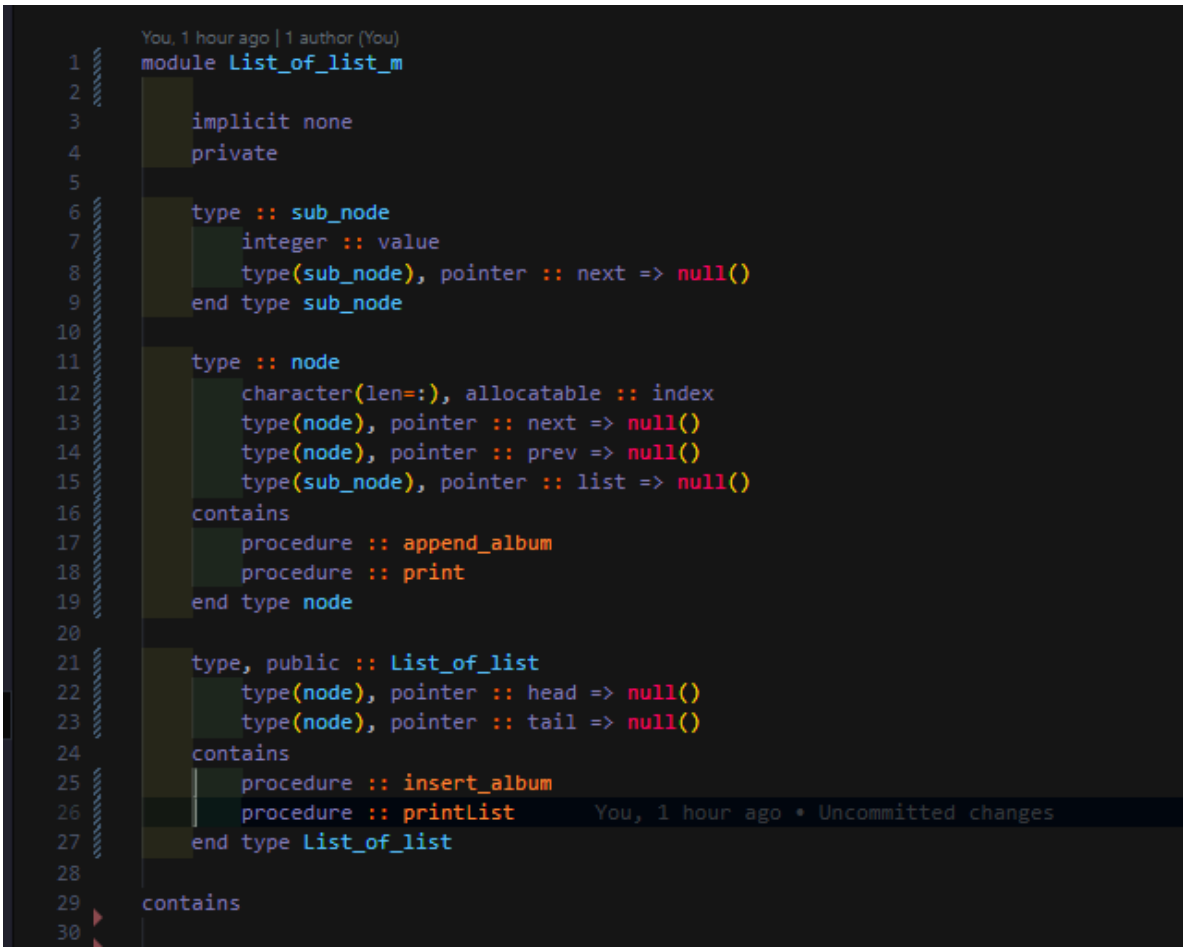
Linked list: guarda la lista para las imágenes de los álbumes

```
module linked_list_m
    implicit none
    private

    public :: user
    public :: linked_list
    public :: push_usuario, append_usuario, print_users, destructor, get_head

    type :: user


        character(len=20) :: nombre_completo
        character(len=20) :: dpi
        character(len=20) :: contrasena
        type(user), pointer :: next => null()
    end type user

    type, public :: linked_list
        private
        type(user), pointer :: head => null()

    contains
        procedure :: push_usuario
        procedure :: append_usuario
        ! procedure :: insert
        procedure :: print_users
        final :: destructor
contains
    subroutine push_usuario(self, nombre_completo, dpi, contrasena)
        class(linked_list), intent(inout) :: self
        character(len=20), intent(in) :: nombre_completo
        character(len=20), intent(in) :: dpi
        character(len=20), intent(in) :: contrasena

        type(user), pointer :: new_user
        allocate(new_user)

        new_user%nombre_completo = nombre_completo
        new_user%dpi = dpi
        new_user%contrasena = contrasena

        if(.not. associated(self%head)) then
            self%head => new_user
        else
            new_user%next => self%head
            self%head => new_user
        end if
    end subroutine push_usuario
```

```
subroutine append_usuario(self, nombre_completo, dpi, contrasena)
    class(linked_list), intent(inout) :: self
    character(len=*), intent(in) :: nombre_completo
    character(len=*), intent(in) :: dpi
    character(len=*), intent(in) :: contrasena

    type(user), pointer :: current_user
    type(user), pointer :: new_user
    allocate(new_user)

    new_user%nombre_completo = nombre_completo
    new_user%dpi = dpi
    new_user%contrasena = contrasena

    if(.not. associated(self%head)) then
        self%head => new_user
    else
        current_user => self%head
        do while(associated(current_user%next))
            current_user => current_user%next
        end do

        current_user%next => new_user
    end if

end subroutine append_usuario
```

```
contains
    subroutine print_users(self)
        class(linked_list), intent(in) :: self
        type(user), pointer :: current_user
        current_user => self%head

        do while(associated(current_user))
            print *, "Nombre completo: ", trim(current_user%nombre_c
            print *, "DPI: ", trim(current_user%dpi)
            print *, "Contraseña: ", trim(current_user%contrasena)
            print *
            current_user => current_user%next
        end do
    end subroutine print_users

    subroutine destructor(self)
        type(linked_list), intent(inout) :: self
        type(user), pointer :: aux_user

        do while(associated(self%head))
            aux_user => self%head%next
            deallocate(self%head)
            self%head = aux_user
        end do
    end subroutine destructor

    function get_head(self) result(head_ptr)
        class(linked_list), intent(in) :: self
        type(user), pointer :: head_ptr

        head_ptr => self%head
    end function get_head

end module linked_list_m
```

Listo f list: guarda las listas de imágenes en una lista doblemente enlzada circular.



```
You, 1 hour ago | 1 author (You)
module List_of_list_m

    implicit none
    private

    type :: sub_node
        integer :: value
        type(sub_node), pointer :: next => null()
    end type sub_node

    type :: node
        character(len=:), allocatable :: index
        type(node), pointer :: next => null()
        type(node), pointer :: prev => null()
        type(sub_node), pointer :: list => null()
    contains
        procedure :: append_album
        procedure :: print
    end type node

    type, public :: List_of_list
        type(node), pointer :: head => null()
        type(node), pointer :: tail => null()
    contains
        procedure :: insert_album
        procedure :: printList      You, 1 hour ago • Uncommitted changes
    end type List_of_list

contains

```

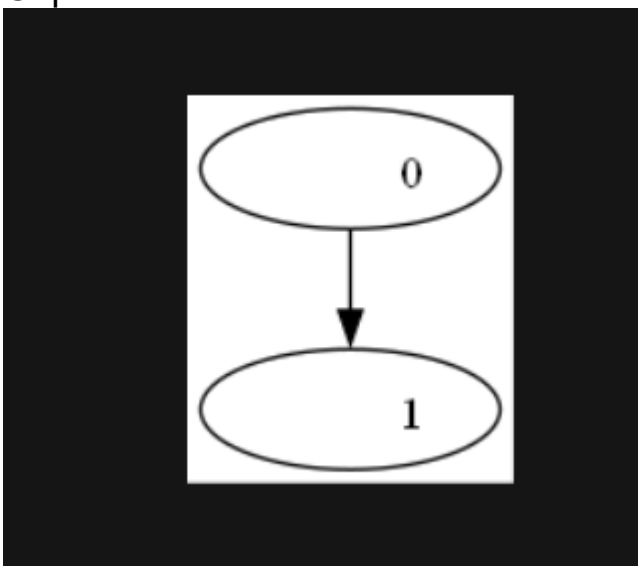
```
subroutine insert_album(self, index, value)
    class(List_of_list), intent(inout) :: self
    integer, intent(in) :: value
    character(len=*), intent(in) :: index

    type(node), pointer :: aux
    type(node), pointer :: new
    allocate(new)

    if(.not. associated(self%head)) then
        allocate(aux)
        aux%index = index
        self%head => aux
        self%tail => aux
        call aux%append_album(value)
    else
        if(index < self%head%index) then
            self%head%prev => new
            new%next => self%head
            self%head => new

            new%index = index
            call new%append_album(value)
        else
            aux => self%head
            do while (associated(aux%next))
                if(index < aux%next%index) then
                    if(index == aux%index) then
                        call aux%append_album(value)
                    else
                        new%next => aux%next
                        new%prev => aux
                        aux%next%prev => new
                        aux%next => new

                        new%index = index
                        call new%append_album(value)
                    end if
                    return
                end if
                aux => aux%next
            end do

            if(index == aux%index) then
                call aux%append_album(value)
            else
                self%tail%next => new
                new%prev => self%tail
                self%tail => new

                new%index = index
```

```
    end subroutine insert_album

    subroutine printList(self)
        class(List_of_list) :: self
        type(node), pointer :: aux

        aux => self%head

        do while(associated(aux))
            print *, 'Indice: ', aux%index
            call aux%print()
            print *, ""
            aux => aux%next
        end do
    end subroutine printList

    !Subrutina y funciones de sub nodo
    subroutine append_album(self, value)
        class(node), intent(inout) :: self
        integer, intent(in) :: value

        type(sub_node), pointer :: aux
        type(sub_node), pointer :: new

        allocate(new)
        new%value = value

        if(.not. associated(self%list)) then
            self%list => new
        else
            aux => self%list
            do while(associated(aux%next))
                aux => aux%next
            end do
            aux%next => new
        end if
    end subroutine append_album

    subroutine print(self)
        class(node), intent(inout) :: self

        type(sub_node), pointer :: aux
        aux => self%list

        do while(associated(aux))
            print *, aux%value
            aux => aux%next
        end do
    end subroutine print
end module List_of_list_m
```

Imágenes esperadas:

Árbol b:

Capas



Matriz